Habilitation à Diriger les Recherches
mention Informatique
Université de Rennes 1

Bayesian Networks of Dynamic Systems

(Version du 28 Mars 2007)

Eric Fabre

Soutenue le 14 juin 2007, devant le jury composé de

Président	**Michel Raynal**, Prof. d'Informatique, Univ. Rennes 1
Rapporteurs	**Stéphane Lafortune**, Prof. of EECS, Univ. of Michigan **Alan Willsky**, Prof. of EE, MIT, Cambridge (US) **Glynn Winskel**, Prof. of CS, Univ. of Cambridge (UK)
Examinateurs	**Albert Benveniste**, DR INRIA, IRISA, Rennes **Christophe Dousson**, Resp. d'Eq. de Rech., France Telecom R&D **Alessandro Giua**, Prof. of EE, Univ. de Cagliari

IRISA/INRIA
Campus de Beaulieu
35042 Rennes cedex
Eric.Fabre@irisa.fr

Contents

Chapter 1

Introduction

> *"(...) diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre.*
>
> *(...) conduire par ordre mes pensées, en commençant par les objets les plus simples et les plus aisés à connaître, pour monter peu à peu, comme par degrés, jusques à la connaissance des plus composés ; et supposant même de l'ordre entre ceux qui ne se précèdent point naturellement les uns les autres."*
>
> René Descartes, in *Discours de la méthode*[1] (1637)

1.1 Motivation

The decomposition principle splitting a complex task into simpler subtasks, has founded decades of technological achievements. Most systems we commonly use every day involve complex chains of reactions between components of different scales, that combine to form the service we expect from them. Assembling components to form a more powerful system is so efficient and widespread that it almost forms a commonplace to mention it. However, considering with more attention the complexity levels of current systems, and the way they are designed today, suggests that this paradigm has reached some limitations.

1. **Size.** The most obvious limitation : The explosive number of elements involved in some applications now reaches complexity levels that go far beyond what a single person can master.

2. **Historical constraints.** Very often, redesign from scratch has become intractable or simply too expensive, or is impossible for matters of downward

[1] Among the four principles of the method, these are the 2nd and 3rd ones. In substance, 1/ take nothing for granted, 2/ divide a complex problem into simpler sub-problems, 3/ understand each elementary sub-problem, and recombine, 4/ don't forget anything ! Summarized as "divide and conquer" by fast readers.

compatibility. So one is bound to continuously upgrade parts of an existing system or software, which leads to a discrepancy between recent and old technologies, recent and old design paradigms.

3. **Heterogeneity.** In the same way, there is also a trend to rapidly assemble off-the-shelf components, of different generations and manufacturers, in order to follow the market demand or new offers proposed by competitors. This produces systems that sometimes have unexpected behaviors, whence the necessity of heavy test procedures. As a matter of fact, most softwares or services today are delivered in a continuous flow of versions, following a continuous flow of bug reports.

4. **Open systems.** In many cases, compound systems are no longer closed systems, like a chip, a computer or a plane, for which the manufacturer could theoretically master all components. They are rather open systems, only partially known to people in charge of their monitoring. In particular in the field of telecommunication networks, or of distributed softwares.

5. **Unstructured systems.** After decades of hierarchical decomposition into components, new design paradigms appear under the form of peer architectures, where components are both clients and servers of one another. Much less intuitive objects in terms of management.

6. **Dynamic architectures.** Moreover, the structure itself of some current software systems is no longer stable and designed once for all, but may be built "on demand." This was partly the case in telecommunication networks, but it becomes a central feature in peer-to-peer networks or in web services, two fast growing application paradigms.

One may be happy with this situation, as far as the main function of a complex system is globally satisfied, and continuously improved by a "test and modify" process. But of course, this can't be sufficient for critical applications. New tools and concepts are continuously needed to assess beforehand whether a system fulfills its objectives or not, whether it is error-free or not. And once a large system has been deployed, similar difficulties remain at run time to monitor it and analyze its behavior, which is the topic of this document.

As a typical domain where these issues are becoming of critical importance, let us mention telecommunication networks and services management. It is by now considered that network elements (NE) are so complex and offer so many features and adjustable parameters, meanwhile networks increase in size and heterogeneity of equipment and functions, that the traditional monitoring of a network by directly accessing and tuning NEs has become impossible. It is commonly considered that a human operator needs one year to master a new NE technology and be able to parameterize the network he supervises. Research groups like the NMRG (Network Management Research Group) at IRTF (Internet Research Task Force), or the European research network EMANICS (MANagement of Internet technologies and Complex Services), are now orienting research to new management concepts. Under

the generic name of *autonomic communications,* objectives like self-configuration, self-healing, self-adaptation, self-xxx reveal a trend to abstract the inherent complexity of systems and search for high-level programming mechanisms for telecommunication networks. Ideally, one should be able to program a network by assigning it service-level objectives. What technologies will bridge the gap between these high-level objectives and low-level management is still unclear and remains a very active research field. One trend is to push down these high-level requirements, under the form of *policy-based management* (essentially for performance management). Another strong tendency favors probabilistic methods, both to model network behaviors (performance and configuration management), or to understand its behavior (learning methods or statistical techniques for fault management and event correlation).

The research work we summarize in this document addresses complexity issues in the reverse direction. We start from traditional model-based approaches to some monitoring problems, that were successful so far for small size systems, and propose a methodology to extend them to possibly large networks of components. The idea is quite simple : the complexity of networked systems precisely comes from the existence of many interconnected functions and elements. Why not turning this to our advantage and imagining a monitoring architecture that would itself be distributed ? Ideally, this would both solve scalability issues, and allow a natural upgrade of monitoring architectures as the structure of the supervised system is updated. As a matter of fact, we'll see that our approach naturally leads to the fashionable idea of peer to peer monitoring architectures, but with a *sound algebraic basis*.

Most of this research was motivated by a target application : failure diagnosis in telecommunication networks. The results we obtained have been successfully implemented and tested on different network technologies, in cooperation with industrial partners[2]. But beyond these direct applications, the theory seems rich and promising enough to address some of the difficulties mentioned above. For example systems with a varying structure, like Web Services. Very likely also, off-line problems like model checking for large components can be addressed with this approach. In summary, between small systems that can be studied as a whole, and large ones that can only be built and tested, or modeled with probabilistic methods, there seems to be some accessible land to explore.

1.2 A distributed approach to monitoring problems

We focus on discrete event dynamic systems (DEDS), and in particular on distributed systems obtained by assembling a large number of components, that we could also call *networks of dynamic systems*. Such systems very rapidly become intractable as their size augments. This is due to combinatorial explosions that take place both in their state space, and in their trajectory space. Because of these com-

[2]Experiments have been carried out for SDH optical network, MPLS networks, submarine line terminal equipment, and GSM radio access network. Alcatel R&I is currently evaluating the introduction of this technology into ALMAP, its corporate network management platform.

binatorial explosions, global (or centralized) approaches developed for monitoring problems are no longer applicable. By "monitoring problem," we encompass problems like supervisory control, optimal control, optimal state/trajectory estimation, or diagnosis problems. Several authors have proposed to address the challenge of distributed systems by means of distributed (or modular) methods. The central idea is to solve the target monitoring problem by parts, at the scale of a single component, in such a way that combining local/partial solutions gives the global one. Specifically, there exist two strategies to do so[3] :

- In the *decentralized* monitoring architecture, a local supervisor is attached to each component (or group of), and has only a *local knowledge*: it only knows the model of that component, plus interface information with the rest of the system, and only has access to observations/measurements coming from that component. Local supervisors perform some computations and forward their results to a coordinator in charge of assembling them. This coordinator is supposed to ignore everything about the supervised system, and has minimal computation capabilities, which means that most of the work is performed by local supervisors.

- In the *distributed* architecture, one is not so much interested in computing a global solution to the monitoring problem, like a global diagnosis, estimates of global states, etc. On the contrary, only *local views* of these global solutions are of interest, that is their projections on each component. Therefore the coordinator becomes useless. The monitoring architecture simplifies into a collection of local supervisors, one per component, having local knowledge and coordinating their work with supervisors of neighboring components to provide a set of coherent local views.

The methodology we propose belongs to the second class, which can be considered as a generalization of the first one, where the necessity of a coordinator is relaxed. The advantage is obvious in terms of scalability : each time a new component is incorporated or replaced into the system, one only has to connect/replace its corresponding local supervisor to upgrade the monitoring architecture (Fig. 1.1). The fact that the connectivity of the supervising architecture must be isomorphic to the interaction structure between components in the system is not casual and will be commented in the next chapters. Notice also that although the knowledge about the "global solutions" to the monitoring problem is distributed in this approach, the information is nevertheless present and available for standard post-processings (like result report, action decision, etc.).

At this point, we made no distinction between *modular* and *distributed* processings. The modularity refers to a problem that can be solved by parts, where each computation "module" is based on a limited knowledge. Typically, computations

[3]In this classification, we omit contributions that do not take into account the *modularity* of the supervised system. For example approaches where several sensors collect different observations on a unique component. Although these approaches are interesting in terms of cooperation between sensors, the modular processing is generally as complex as a global processing based on all observations. Our goal here is to reduce complexity, in order to capture large systems.

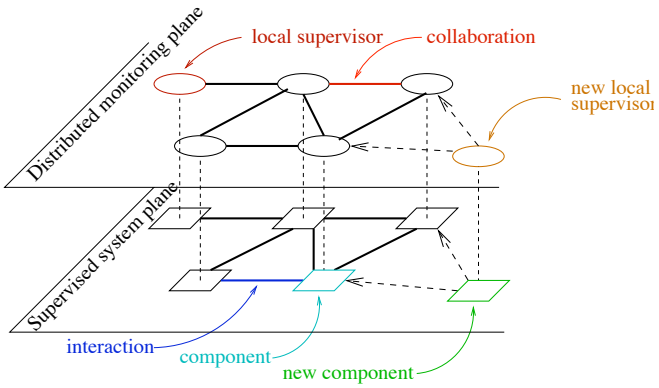


Figure 1.1: *A network of dynamic systems, and its distributed monitoring architecture.*

performed by a local supervisor, knowing only one component and observations that it produced. The expression "distributed processing" goes further by assuming that the partial computations are performed at different locations, and thus require communications between modules. This introduces scheduling issues in the problem: one first has to determine what should be computed locally, what to communicate to neighbors, but also when communication should take place, and what to do with delayed (or lost) messages. These protocol concerns are very sensitive in some approaches [90]. In the framework presented here, we will consider asynchronous distributed systems, and the distributed monitoring algorithms will also be completely asynchronous. Therefore modularity will be equivalent to distribution, and we shall not distinguish them.

1.3 Overview of our contribution

As suggested by the title, this document describes an attempt at assembling two disconnected sets of results, developed in different communities and with apparently unrelated objectives.

Bayesian networks. The term *Bayesian Network*[4] refers to graphical models displaying the correlation structure of a collection of random variables. Let $\mathcal{V} = \{V_k, 1 \leq k \leq K\}$ be a finite set of variables, and let $\mathcal{V} = \mathcal{V}_1 \cup \ldots \cup \mathcal{V}_N$ be a covering of $\mathcal{V}$, with $\mathcal{V}_n \subseteq \mathcal{V}, 1 \leq n \leq N$. We denote by $\mathbf{v}$ a function associating to each variable in $\mathcal{V}$ a value of its domain, and by $\mathbf{v}_i$ we denote the restriction of $\mathbf{v}$ to $\mathcal{V}_n$. We also denote v_k a value of variable V_k. A joint distribution $\mathbb{P}_\mathcal{V}$ on variables $\mathcal{V}$ can be specified in terms of so-called *potential functions* ϕ_n defined on the $\mathbf{v}_n$ and taking values in $\mathbb{R}$. Specifically

$$\mathbb{P}(\mathbf{v}) \quad = \quad \frac{1}{Z} \exp\{-\sum_{n=1}^{N} \phi_n(\mathbf{v}_n)\} \tag{1.1}$$

[4]Sometimes also called Markov random field, graphical model, or belief network, according to the community using it.

where Z is a normalizing factor. Each subset $\mathcal{V}_n$ is called a *clique*. Intuitively ϕ_n defines (soft) constraints on the elements of clique $\mathcal{V}_n$, and by suitably combining all these local constraints, one specifies the global correlation structure in $\mathcal{V}$. To prepare the analogy with dynamic systems, we call the pair $\mathcal{S}_n = (\mathcal{V}_n, \phi_n)$ a *component*.

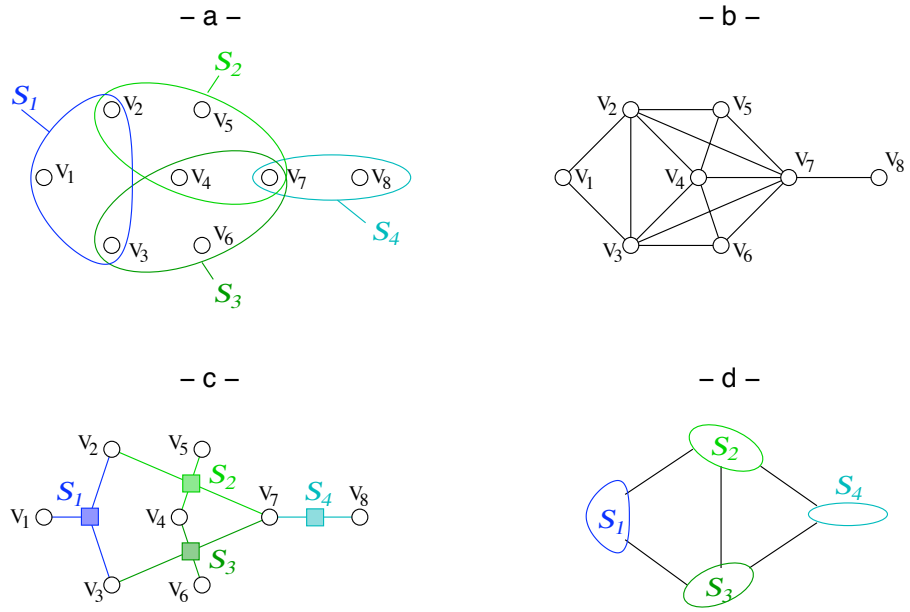


Figure 1.2: *Four graphical representations of dependencies between variables V_k and/or components $\mathcal{S}_n$.*

The interactions in $\mathcal{V}$ admit several graphical representations.

- The most direct : as a hypergraph $H = (\mathcal{V}, \{\mathcal{V}_n\}_{1 \leq n \leq N})$. Every variable in $\mathcal{V}$ is a node, and each subset $\mathcal{V}_n$ defines a hyper-edge (Fig. 1.2-a).
- As a graph $G = (\mathcal{V}, E)$, still with variables as nodes. Two variables are related by an edge of E iff they appear in the same $\mathcal{V}_n$, for some n. Or equivalently, $\mathcal{G}$ restricted to $\mathcal{V}_n$ is a complete graph (Fig. 1.2-b).
- As a bipartite graph $G = (\mathcal{V}, S, E)$. One still has variable $\mathcal{V}$ as first set of nodes, and the second one $S = \{1, \ldots, N\}$ corresponds to the N "systems" defined by the $\mathcal{V}_n$. A variable $V \in \mathcal{V}$ is related by an edge to system n iff $V \in \mathcal{V}_n$ (Fig. 1.2-c).
- As a dual graph $G = (S, E)$ where nodes in $S = \{1, \ldots, N\}$ still represent the $\mathcal{V}_n$. There is an edge in E between n and m iff $\mathcal{V}_n \cap \mathcal{V}_m \neq \emptyset$ (Fig. 1.2-d).

These graphical representations have two main advantages.

1. First of all, they can be directly interpreted in terms of conditional independence statements. Consider Fig. 1.2-b for example. Variables $\{V_4, V_7\}$ *separate* $\{V_1, V_2, V_3, V_5, V_6\}$ from $\{V_8\}$ in the sense that removing V_4 and V_7 from

the graph disconnects these two sets. This property immediately entails that $(V_1, V_2, V_3, V_5, V_6)$ and V_8 are conditionally independent given (V_4, V_7) for $\mathbb{P}$:

$$\mathbb{P}(\mathbf{v}) \quad = \quad \mathbb{P}(v_1, v_2, v_3, v_5, v_6 | v_4, v_7)\, \mathbb{P}(v_8 | v_4, v_7)\, \mathbb{P}(v_4, v_7) \tag{1.2}$$

as it can be checked directly from (1.1). The graph is thus a summary of a set of conditional independence relations. Whence the name "Bayesian network :" (1.2) is a Bayes formula.

2. Secondly, precisely because of these conditional independence relations, *estimation problems* can be resolved by parts. These problems typically take the following form : one observes the value of some variables in $\mathcal{V}$ and wishes to determine the most likely value of all the others. In the simplest cases, *i.e.* when the interaction graph of fig. 1.2-d is a tree, the resolution takes the form of message passing algorithms (MPA), where some computations are performed at the scale of a single clique $\mathcal{V}_n$, and where neighboring cliques exchange messages. The most famous examples of such algorithms appear for Markov chains, *i.e.* Bayesian networks where the cliques are organized in a single string : the Viterbi algorithm, the soft output Viterbi algorithm (SOVA), the Kalman filter, the Rough-Tung-Striebel algorithm, the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm, the forward-backward algorithm, the [min,max]-[sum,product] algorithm, the sum-product algorithm, dynamic programming, the belief propagation are all examples of MPA[5]. For more complex graphs, MPA can still be applied. They are theoretically suboptimal, but yield excellent results in practice, as it was revealed by the iterative algorithms for decoding turbo-codes.

The reader will have noticed that message passing algorithms are a form of distributed processings. We are precisely going to elaborate on this remark.

Networks of dynamic systems. The simplest model of a discrete event dynamic system (DEDS) takes the form of an automaton $\mathcal{A} = (Q, T, v_0)$. $\mathcal{A}$ is composed of a single state variable V taking values in the finite set Q of possible *states*, and initialized at $v_0 \in Q$. $T \subseteq Q \times Q$ is a finite set of *transitions* : a transition $t = (v, v') \in T$ can fire when V takes value v. After the firing, $\mathcal{A}$ is in state $V = v'$. A run of $\mathcal{A}$ is thus a sequence $\sigma = v_0[t_1\rangle v_1[t_2\rangle v_2 \dots v_{l-1}[t_l\rangle v_l \dots$ such that $t_l = (v_{l-1}, v_l) \in T$.

With a very simple idea, this class of DEDS can be extended to encompass much more complex systems : instead of a single variable V, we can define automata operating on *several* state variables. Specifically, we define a *tile system* $\mathcal{S}$ as a triple $\mathcal{S} = (\mathcal{V}, \mathcal{T}, \mathbf{v}^0)$, where $\mathcal{V} = \{V_k, 1 \leq k \leq K\}$ is a set of variables with finite domains, $\mathcal{T}$ is a finite set of *tiles*, and $\mathbf{v}^0$ is the initial "state" of the system. Apparently, we only introduced a vector-valued state variable. The originality comes from the fact that transitions of $\mathcal{T}$, that we call *tiles*, do not operate on all variables at a time. A tile $t = (\mathcal{V}_t, \mathbf{v}_t^-, \mathbf{v}_t^+) \in T$ modifies only variables in $\mathcal{V}_t \subseteq \mathcal{V}$. Specifically, t can

[5]Often the same algorithm appears with different names, according to the community that (re)discovered it !

fire when the state $\mathbf{v}$ of the system restricted to $\mathcal{V}_t$ takes value $\mathbf{v}_t^-$. After the firing, these variables are changed to value $\mathbf{v}_t^+$, and variables in $\mathcal{V} \setminus \mathcal{V}_t$ remain unchanged (fig. 1.3). This formalism is very convenient to define large systems, with numerous state variables, by local dynamics.

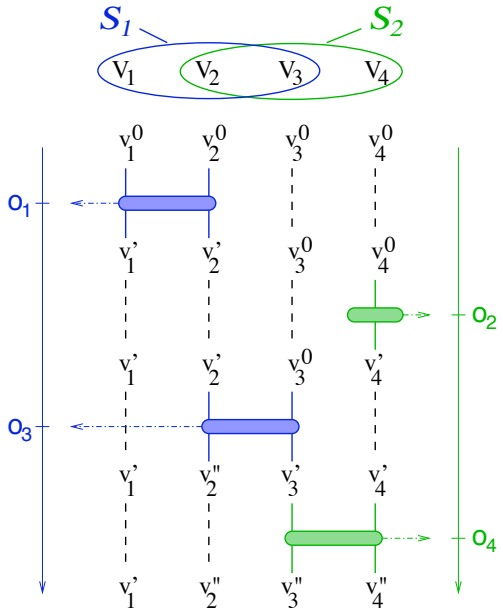


Figure 1.3: *A run of a tile system made of two components $\mathcal{S}_1$ and $\mathcal{S}_2$, that share two variables, V_2 and V_3. The first tile firing in this run (top left) changes only the value of V_1 and V_2, and produces the label o_1.*

The main advantage of this framework is that tile systems can be composed very naturally. There exists several ways to define the composition, that we shall recall in this document. The simplest one, at this point, is the following : let the $\mathcal{S}_n = (\mathcal{V}_n, \mathcal{T}_n, \mathbf{v}_n^0)$ be tile systems, $1 \leq n \leq N$, their composition $\mathcal{S} = (\mathcal{V}, \mathcal{T}, \mathbf{v}^0) = \mathcal{S}_1 \| \mathcal{S}_2 \| \dots \| \mathcal{S}_N$ is obtained by taking the union of variables $\mathcal{V} = \mathcal{V}_1 \cup \dots \cup \mathcal{V}_N$ and of tiles $\mathcal{T} = \mathcal{T}_1 \cup \dots \cup \mathcal{T}_N$, and by assembling the initial states $\mathbf{v}_i^0$ (provided they coincide on shared variables). The interactions come from the fact that two components $\mathcal{S}_n, \mathcal{S}_m$ can *both* read and change the values of the state variables in the intersection $\mathcal{V}_n \cap \mathcal{V}_m$. The latter thus behave as communication ports.

As for Bayesian networks, one can associate graphical representations to a compound tile system $\mathcal{S}$, and they turn out to be exactly those of Fig. 1.2 ! Instead of a potential function ϕ_n defining constraints on a subset $\mathcal{V}_n$ of variables, on has a component $\mathcal{S}_n$ defining local dynamics on the $\mathcal{V}_n$. But we need one more step to make this analogy operational.

The join. In its simplest form, a monitoring problem for $\mathcal{S}$ could be expressed as follows. Assume some of the tiles in the $\mathcal{S}_n$ can emit a possibly random signal when they fire (the production of an alarm for example) that we call a *label*. $\mathcal{S}$ performs a hidden run κ, and the labels emitted by tiles in this run are collected under the form of observations $\mathcal{O}^b$ (Fig. 1.3). Since κ is hidden, the goal is to recover all runs

of $\mathcal{S}$ that could have produced $\mathcal{O}^b$. Further, if each component $\mathcal{S}_n$ is a stochastic system, one would like to recover the most likely run, as an estimate of κ.

This problem looks very much like a standard Hidden Markov Model (HMM) problem. But here, we complexify it a little.

- First we assume that observations $\mathcal{O}^b$ are not collected into a single sequence of labels. Rather, labels are collected on each component $\mathcal{S}_n$ by a local sensor, that produces the local observation set $\mathcal{O}^b_n$. Our observation is thus the tuple $\mathcal{O}^b = (\mathcal{O}^b_1, \mathcal{O}^b_2, \ldots, \mathcal{O}^b_N)$, and the interleaving of events in the $\mathcal{O}^b_i$ is assumed to be lost.

- Secondly, as mentioned in the previous section, we aim at possibly large systems. Therefore the usual procedures for HMMs, that operate on the state space of $\mathcal{S}$, are just unaffordable : the state space size explodes with the number of components. We rather look for a modular methodology to solve the monitoring problem, where computations would be performed at the scale of components $\mathcal{S}_n$.

- Finally, we would like to perform this monitoring on-line, *i.e.* we wish to update our estimates of the hidden run κ on the fly, as new observations are collected on the different sensors.

This is close enough to the MPA we sketched for Bayesian networks. To establish the connection, one must realize that the objects we are interested in are not so much the components $\mathcal{S}_n$ themselves, but rather their sets of runs. So we must introduce time in our formalism. As illustrated in Fig. (1.3), a run κ of $\mathcal{S}$ can be considered as a tuple of trajectories (*i.e.* sequences of events), one per variable V_k. So let us denote by $\bar{V}_k$ a variable whose values are *trajectories* of V_k, for $V_k \in \mathcal{V}$. We are going to use the fact that variables $(\bar{V}_k)_{1\leq k\leq K}$ form a "Markov field," in a very specific sense. Consider an operator $\mathcal{U}$ that would take a tile system $\mathcal{S}$ and compute in some form or another the set $\mathcal{U}(\mathcal{S})$ of all its runs κ. We will show in this document that $\mathcal{U}$ can be designed so as to be a *product preserving functor* on tile systems. In other words, one has

$$\mathcal{U}(\mathcal{S}_1 \| \ldots \| \mathcal{S}_N) \quad = \quad \mathcal{U}(\mathcal{S}_1) \wedge \ldots \wedge \mathcal{U}(\mathcal{S}_N) \tag{1.3}$$

where $\wedge$ is an appropriate composition operator on trajectory sets.

This *factorization property* is the counterpart of (1.1) for networks of dynamic systems, and is at the core of the methodology we present in this document. Each $\mathcal{U}(\mathcal{S}_n)$ can be considered as a "potential function" or as the definition of local constraints on variables $\bar{\mathcal{V}}_n = \{\bar{V}, V \in \mathcal{V}_n\}$. Because of (1.3), the interaction structure of the $\mathcal{U}(\mathcal{S}_n)$ is *identical* to that of the components $\mathcal{S}_n$ themselves. Finally, observations $\mathcal{O}^b_n$ as well can be interpreted as some knowledge on the local trajectories in $\mathcal{U}(\mathcal{S}_n)$, *i.e.* on the values of variables in $\bar{\mathcal{V}}_n$. *So we are almost back to a static problem that can be solved by MPA.*

This forms the essential message of this document : **many results obtained for Bayesian networks can be recycled into distributed and asynchronous estimation algorithms for distributed dynamic systems**.

1.4 Organization of the document

The next chapter describes an axiomatic framework designed to capture both Bayesian networks and networks of dynamic systems. The objective is to describe message passing algorithms (MPA) in a formalism that encapsulates both situations. We define abstract systems operating on variables, that we compose by shared variables. Only two operators are useful on these systems : a composition and a projection (or reduction). We relate them by a small set of axioms, from which many algebraic properties can be derived. The interaction structure of a compound system can be described by a graph, on which the standard separation criterion is equivalent to a form of conditional independence. This is sufficient to develop MPA, study their convergence and explore the properties of their stationary points.

Chapter 3 is a first application of this framework to dynamic systems, in the simplest possible setting. We define a network of dynamic systems as the composition of automata by the usual parallel product. This composition is slightly modified to keep track of components when they are assembled. Such systems are provided with the usual sequential semantics : their runs are simply sequences of events. The distributed diagnosis problem is defined in this setting, and solved in different ways. First of all we reason on languages, which highlights the architecture of the computations that we apply all along this document. But languages are very inefficient to encode large sets of runs of a system, so they are inappropriate to on-line monitoring algorithms. The notion of trellis process is then introduced to describe sets of runs in a compact manner. We show that these objects enjoy a nice factorization property and satisfy the axiomatic framework of chapter 2, which allows us to compute with them. The key point here,*i.e.* the factorization property of trellis processes, is derived by category theory arguments. We conclude this chapter by showing some drawbacks of the sequential semantics, and advocate true concurrency semantics to deal with distributed systems.

Chapter 4 proposes a first setting to handle sets of runs in the true concurrency semantics. We first change the notion of composition in order to preserve the state variables of components, rather that merging them in a big product state variable. As in the case of networks of automata, composition can be done either by product or by shared components : both situations are equivalent. The notion of system this leads to turns out to be almost equivalent to safe Petri nets. In the true concurrency semantics, runs are partial orders of events (or Mazurkiewicz traces), and sets of runs can be encoded under the form of branching processes. The latter enjoy once again a nice factorization property (still derived by category theory arguments), and admit a natural notion of projection. However, the axioms of chapter 2 are satisfied only in very specific cases. To perform computations in the general case, one must introduce the notion of augmented branching process.

Chapter 5 tries to go further and explores the existence of trellis processes for the true concurrency semantics, as an even more compact way of encoding sets of runs. Still in view of efficient distributed and on-line monitoring algorithms. Trellis processes can indeed be defined, and enjoy very elegant properties. In particular, the factorization property is preserved, and a natural notion of projection exists, which

allows us once again to apply the formalism of chapter 2 in some specific cases. But the notion of augmented trellis process, that would encompass the general case, is still missing.

Chapter 6 gives some snapshots at different contracts that guided this research. It describes the applications that were considered and some features of the proposed solutions. It underlines in particular how the implementations that were experimented were either in advance, or late, or sometimes erroneous with respect to the development of the corresponding theory.

As a conclusion, chapter 7 summarizes the state of this theory and identifies some missing tiles in the puzzle. It also lists a number of immediate or more futuristic extensions of this work.

1.5 Historical perspective

The assembling of Bayesian networks, distributed dynamic systems and category theory presented in this document doesn't yet form a completely smooth theory. But it already went through several polishing phases. We briefly mention some of them to underline the benefits of crossing different scientific cultures, but also to show how simple ideas sometimes take complex ways to materialize, ways in which chance plays an important part.

The origins. The problem that triggered this research was jointly raised in 1996 by Claude Jard (background in distributed programming) and Albert Benveniste (several backgrounds, in particular signal processing and random processes). In telecommunication networks, failures generally propagate in the net which causes bursts of alarms at different locations. These alarms are only partially ordered in time, due to the distributed nature of the network. So the original problem was to identify failures from patterns of partially ordered alarms. The idea of interacting components and of stochastic systems were also present at the very beginning, which oriented us to Bayesian networks.

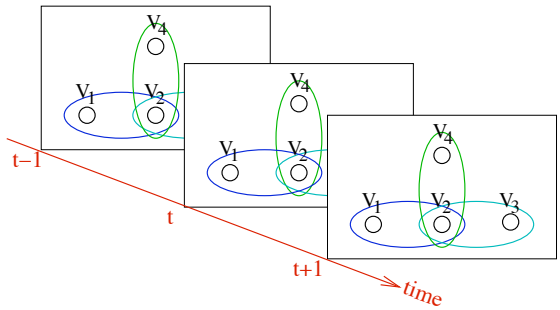


Figure 1.4: *Augmenting interactions in space with interactions in time.*

Very soon however, we were faced with the difficulty of introducing time, or dynamics, in Bayesian networks. When there is a single state variable, for example

in Markov chains, this is done by duplicating the variable to represent its value at each clock tick. Time is unfolded, in some sense. And the Markov chain dynamics is introduced by potential functions coupling variables at time t and $t+1$. Applied to a Bayesian network, this principle would amount to add one more dimension to Fig. 1.2, perpendicular to the page, that would represent the time axis (Fig. 1.4). But in a distributed system, time is not homogeneous for all variables : as illustrated in Fig. 1.3, some variables may evolve while others remain constant. And the places where transitions occur depend the run ! To capture this unusual feature, one would need a Bayesian network (or a Markov random field) whose structure would depend on the value of some of its variables. Unfortunately, such a theory doesn't exist yet, and seems difficult to conceive.

Petri nets. It was thus chosen to abandon the idea of a global stochastic model capturing both interactions in space and in time. The first framework was based on partially stochastic safe Petri nets. Petri nets are a natural model for concurrent systems : they describe well the idea of local transitions, and the true concurrency semantics allows us to describe their runs as partial orders of events, also called *configurations*. The term "partially stochastic" relates to the fact that transitions have a "weight," related to their likelihood, which allows us to compare runs. However no proper probabilistic space of runs was derived. This setting was sufficient to perform a Viterbi-like algorithm, and recover the "most likely" trajectory explaining a sequence (or a tuple of sequences) of alarms.

Estimation algorithms were soon extended into distributed procedures, for nets with several components connected by shared places. The key observation was that configurations of the global net could be split into local configurations of its components. The resulting algorithm took the form of several cooperating Viterbi algorithms : one per component, in charge of recovering configurations of this component, and proposing to neighbors its possible explanations for shared places. After all observations were processed, a belief propagation phase was initiated to find the best combination of local explanations.

Unfoldings. Handling sets of configurations can be quite heavy since these objects rapidly become large. To minimize the memory space, configurations were encoded with a back-pointer notation, as in [35, 31]. In other words, the underlying data structure we used was the *unfolding* of the system, and a configuration was nothing more than a tuple of entry points in this data structure. This revealed that the diagnosis could probably be performed directly in terms of unfoldings, which was done in [9]. Moreover, the distributed version of the algorithm suggested that the unfolding of a compound system factorized into local unfoldings. This was established directly in [37], and distributed estimation algorithms were re-expressed in this setting : messages were now pieces of unfoldings.

Several results were then obtained in parallel. First of all, the derivation of an axiomatic framework to express general message passing algorithms (MPA) and study their algebraic properties [38]. In particular to express conditional independence, and study convergence of MPA. Secondly, the introduction of *augmented branching processes* (ABP), as the correct framework to express distributed algorithms

(ordinary branching processes are not sufficient) [40]. Finally, the expression of distributed algorithms in terms of event structures, instead of branching processes [41].

Factorization properties and category theory. By a strike of fate, while I was exploring different families of event structures to check if ABP already existed, I came aware of Winskel's work on models for concurrency. I was struck in particular by a result in [110], stating in a couple of lines the factorization property (1.3) on unfoldings[6]. The key argument for this derivation was some obscure result in category theory... which I thus decided to investigate, motivated by the tedious proofs in [37]. And by Winskel's killing sentence in [110] :

> *Proving these facts directly from the unfolding construction is quite unwieldy - and completely uninstructive - so it is fortunate there is this abstract characterization of the occurrence net unfolding of a [safe] net. In a sense, it was there all the time, because (...), so it was determined (...) by the categorical set-up.*

The category theory approach brought many advantages, by focusing developments on the essential features, while saving us from tedious and useless proofs. For example, interactions between components can be expressed under the form of synchronous products, or by shared variables (pullbacks [43]), without changing the theory. Several other event structures were also quickly derived, like trellis processes for the true concurrency semantics [45], or trellises for the usual interleaving semantics [44, 42], and their factorization properties came almost for free, thus making them available to distributed algorithms.

Markov nets. These elements of history wouldn't be complete without mentioning the collaboration with Stefan Haar, met at a workshop on Petri nets (GDR ARP[7], June 2000) where I was invited to give a talk about partially stochastic Petri nets. Just like us, Stefan was trying at that time to randomize concurrent systems. The difficulty was to obtain some equivalence between concurrency and stochastic independence. In other words, components that do not interact should also be independent in the stochastic sense, which is not achieved in any form of stochastic Petri nets. Combining our approaches, Albert, Stefan and I managed to randomize unfoldings of a reasonable class of safe nets, and to define a Markov property on unfoldings, based on a notion of stopping time. This resulted in *Markov nets* [39], that was later refined by Samy Abbes who introduced the simpler notion of branching cell (see Samy's thesis [1] and [2]). Notice however that the availability of a genuine stochastic framework is important for identification issues, or performance analysis, but it has little influence on estimation algorithms, where one is only interested in comparing the relative likelihoods of two trajectories ; so a definition "up to a constant" is sufficient.

[6] This paper was handed to me by Samy Abbes, for a completely different purpose.

[7] A CNRS funded research group, dedicated to architectures, networks and systems, and parallelism.

Chapter 2

Graphical models of interactions

The formalism presented in this chapter aims at a double objective. First of all, we want to introduce the minimum amount of concepts allowing us to define graphical models of interactions and message passing algorithms. The idea is that the simpler the formalism, the broader its scope. Secondly, with these simple tools, we want to go as far as possible in terms of distributed algorithms, convergence properties, etc.

We thus start with a simple definition of systems, "operating" on variables, and two operations. The first one allows us to compose systems, the second one allows us to reduce systems to part of their variables. With a simple set of axioms on these two operators, in particular a form of conditional independence property, one can derive interaction graphs of systems and message passing algorithms. In some cases, convergence properties can be established. This formalism is the basis on which distributed algorithms will be built, when we move to networks of dynamic systems.

2.1 Systems and their graphs

Notations. Specifying the notations of the introduction, we consider a finite set $\mathcal{V}_{max}$ of variables. A variable $V \in \mathcal{V}_{max}$ takes values v in domain $\mathcal{D}_V$. Variable sets are denoted with script letters $\mathcal{V} \subseteq \mathcal{V}_{max}$, and bold-face letters like $\mathbf{v}$ represent functions over $\mathcal{V}$, associating a value of $\mathcal{D}_V$ to each variable $V \in \mathcal{V}$. We call $\mathbf{v}$ a *(local) state*, and, for convenience, we sometimes represent it as a tuple of values $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ assuming there exists a natural ordering of variables in $\mathcal{V} = \{V_1, \ldots, V_n\}$, and we denote by $\mathcal{D}_{\mathcal{V}} = \mathcal{D}_{V_1} \times \ldots \times \mathcal{D}_{V_n}$ the domain of values for $\mathbf{v}$.

2.1.1 Systems

We consider an abstract notion of system over these variables, that we generically denote by $\mathcal{S}$. To help intuition, systems can be understood as sets of tuples $\mathbf{v}_{max} \in \mathcal{D}_{\mathcal{V}_{max}}$. Systems are provided with two operations: *composition* and *reduction.* The composition $\mathcal{S} = \mathcal{S}_1 \wedge \mathcal{S}_2$ is associative and commutative. The reduction takes the form of a family of operators $\Pi_{\mathcal{V}}$, indexed by sets of variables $\mathcal{V} \subseteq \mathcal{V}_{max}$, and operating on a single system. Intuitively, $\Pi_{\mathcal{V}}(\mathcal{S})$ "projects" system $\mathcal{S}$ on variables $\mathcal{V}$.

We provide this setting with the following axioms :

$$\forall\, \mathcal{V}_1, \mathcal{V}_2 \subseteq \mathcal{V}_{max}, \qquad \Pi_{\mathcal{V}_1} \circ \Pi_{\mathcal{V}_2} \;=\; \Pi_{\mathcal{V}_1 \cap \mathcal{V}_2} \tag{a1}$$

which expresses that reduction operators are actually projections.

$$\forall \mathcal{S},\; \exists \mathcal{V} \subseteq \mathcal{V}_{max} \;:\; \Pi_{\mathcal{V}}(\mathcal{S}) \;=\; \mathcal{S} \tag{a2}$$

System $\mathcal{S}$ is said to *operate on variables of* $\mathcal{V}$. Using (a1), one can derive the existence of a smaller variable set on which $\mathcal{S}$ operates, denoted by $\mathcal{V}_{\mathcal{S}}$.

The central axiom concerns the relation between composition and reduction. Let $\mathcal{S}_1, \mathcal{S}_2$ be two systems operating respectively on $\mathcal{V}_1, \mathcal{V}_2$, then

$$\forall \mathcal{V}_3 \supseteq \mathcal{V}_1 \cap \mathcal{V}_2, \qquad \Pi_{\mathcal{V}_3}(\mathcal{S}_1 \wedge \mathcal{S}_2) \;=\; \Pi_{\mathcal{V}_3}(\mathcal{S}_1) \wedge \Pi_{\mathcal{V}_3}(\mathcal{S}_2) \tag{a3}$$

(a3) expresses that the interaction between systems $\mathcal{S}_1$ and $\mathcal{S}_2$ is completely captured by their shared variables $\mathcal{V}_1 \cap \mathcal{V}_2$, which thus behave as an interface between the two systems. Notice also the striking similarity of (a3) with the conditional independence statement $\mathbb{P}(\mathcal{V}_1, \mathcal{V}_2|\mathcal{V}_3) = \mathbb{P}(\mathcal{V}_1|\mathcal{V}_3)\mathbb{P}(\mathcal{V}_2|\mathcal{V}_3)$, expressing that $\mathcal{V}_3$ captures all statistical dependencies between $\mathcal{V}_1$ and $\mathcal{V}_2$ for distribution $\mathbb{P}$. It is a well known fact that such independence statements form the basis of recursive estimation algorithms, and we are indeed going to build our algorithms on this property.

To illustrate the power of this axiom, let us replace $\mathcal{S}_i$ by $\Pi_{\mathcal{V}_i}(\mathcal{S}_i)$ on the right hand side of (a3), and apply (a1). One gets

$$\forall \mathcal{V}_3 \supseteq \mathcal{V}_1 \cap \mathcal{V}_2, \qquad \Pi_{\mathcal{V}_3}(\mathcal{S}_1 \wedge \mathcal{S}_2) \;=\; \Pi_{\mathcal{V}_3 \cap \mathcal{V}_1}(\mathcal{S}_1) \wedge \Pi_{\mathcal{V}_3 \cap \mathcal{V}_2}(\mathcal{S}_2) \tag{2.1}$$

Taking $\mathcal{V}_3 = \mathcal{V}_1 \cup \mathcal{V}_2$ in (2.1) yields

$$\begin{aligned} \Pi_{\mathcal{V}_1 \cup \mathcal{V}_2}(\mathcal{S}_1 \wedge \mathcal{S}_2) &= \Pi_{\mathcal{V}_1}(\mathcal{S}_1) \wedge \Pi_{\mathcal{V}_2}(\mathcal{S}_2) \\ &= \mathcal{S}_1 \wedge \mathcal{S}_2 \end{aligned} \tag{2.2}$$

which expresses that $\mathcal{S}_1 \wedge \mathcal{S}_2$ operates on variables of $\mathcal{V}_1 \cup \mathcal{V}_2$, a natural property one could expect from composition.

The last axiom we introduce is essentially technical : it assumes the existence of an identity element $\mathbb{1}$ for composition :

$$\forall \mathcal{S}, \quad \mathcal{S} \wedge \mathbb{1} \;=\; \mathcal{S} \tag{a4}$$

It is also natural to require that $\mathbb{1}$ do not operate on any variable, *i.e.* $\mathcal{V}_{\mathbb{1}} = \emptyset$, or $\Pi_{\emptyset}(\mathbb{1}) = \mathbb{1}$[1]. By (a1), this induces $\Pi_{\mathcal{V}}(\mathbb{1}) = \mathbb{1}$ for all $\mathcal{V} \subseteq \mathcal{V}_{max}$, and so $\mathcal{S} \wedge \Pi_{\mathcal{V}}(\mathbb{1}) = \mathcal{S}$ for all $\mathcal{S}$.

[1] A more elegant property would be $\forall \mathcal{S}, \Pi_{\emptyset}(\mathcal{S}) = \mathbb{1}$, but we actually don't need this stronger assumption.

2.1.2 Examples

Constraint systems. Recall that a local state $\mathbf{v}$ is a function $\mathbf{v} : \mathcal{V} \to \mathcal{D}_{\mathcal{V}}$ with $\mathcal{V} \subseteq \mathcal{V}_{max}$. So $\mathbf{v}$ can be considered as a set of global states where the value is fixed on variables of $\mathcal{V}$ and free on $\bar{\mathcal{V}} = \mathcal{V}_{max} \setminus \mathcal{V}$. More specifically, we define the *span* of $\mathbf{v}$ as the set of all global states $\mathbf{v}_{max}$ obtained by extending $\mathbf{v}$ into total functions over $\mathcal{V}_{max}$, in all possible ways.

We define a constraint system $\mathcal{S}$ as a set of (local) states, not necessarily fixing the value of the same variables, and we say that $\mathbf{v}_{max}$ satisfies (or belongs to) $\mathcal{S}$ if it belongs to the span of $\mathcal{S}$: $Span(\mathcal{S}) \triangleq \cup_{\mathbf{v}\in\mathcal{S}} Span(\mathbf{v})$. Systems with identical spans are considered as equivalent: we don't distinguish them.

Let $\mathcal{V}' \subseteq \mathcal{V}_{max}$, the reduction $\Pi_{\mathcal{V}'}(\mathbf{v})$ of a state $\mathbf{v}$ to $\mathcal{V}'$ simply corresponds to the restriction $\mathbf{v}_{|\mathcal{V}'} : \mathcal{V} \cap \mathcal{V}' \to \mathcal{D}_{\mathcal{V}\cap\mathcal{V}'}$, where only values over $\mathcal{V} \cap \mathcal{V}'$ remain fixed. The reduction of a system follows: $\mathcal{S}' = \Pi_{\mathcal{V}'}(\mathcal{S}) \triangleq \{\Pi_{\mathcal{V}'}(\mathbf{v}) : \mathbf{v} \in \mathcal{S}\}$. Observe that $\mathcal{S}' = \Pi_{\mathcal{V}'}(\mathcal{S}')$, which underlines that $\mathcal{S}'$ specifies constraints on variables of $\mathcal{V}'$ only. In that case, the span of $\mathcal{S}'$ in $\mathcal{D}_{\mathcal{V}_{max}}$ can be uniquely represented as the union of local states of shape $\mathbf{v}' : \mathcal{V}' \to \mathcal{D}_{\mathcal{V}'}$. We adopt notation $(\mathcal{S}', \mathcal{V}')$ to express that $\mathcal{S}'$ operates on $\mathcal{V}'$, and that elements of $\mathcal{S}'$ are in the canonical form $\mathbf{v}'$.

The composition of $\mathcal{S}_1$ and $\mathcal{S}_2$ is defined by the conjunction of their constraints, *i.e.* by the intersection of their spans. In other words, assuming $(\mathcal{S}_i, \mathcal{V}_i)$, the composition of $\mathbf{v}_1$ and $\mathbf{v}_2$ is non empty iff $\mathbf{v}_{1\,|\mathcal{V}_2} = \mathbf{v}_{2\,|\mathcal{V}_1}$. And in that case, the resulting state $\mathbf{v}_1 \wedge \mathbf{v}_2$ is obtained by merging the partial functions $\mathbf{v}_1$ and $\mathbf{v}_2$ into a partial function over $\mathcal{V}_1 \cup \mathcal{V}_2$. The definition of $\mathcal{S}_1 \wedge \mathcal{S}_2$ follows: $\mathcal{S}_1 \wedge \mathcal{S}_2 = \{\mathbf{v}_1 \wedge \mathbf{v}_2 : \mathbf{v}_i \in \mathcal{S}_i\}$.

The unit system $\mathbb{1}$ is defined as the set $\mathcal{D}_{\mathcal{V}_{max}}$, *i.e.* $\mathbb{1}$ allows all possible states. Obviously, $\forall \mathcal{V} \subseteq \mathcal{V}_{max}$, $\Pi_{\mathcal{V}}(\mathbb{1}) = \mathbb{1}$, so $\mathbb{1}$ operates on no variable, and its canonical form reduces to the universal state $*$.

It is straightforward to check that composition and reduction of constraint systems satisfy axioms (a1) to (a4).

Probabilistic systems. This class extends the previous one. For simplicity, we only consider constraint systems $(\mathcal{S}, \mathcal{V})$ in canonical form, that we extend into $(\mathcal{S}, \mathcal{V}, \mathcal{C})$ where $\mathcal{C} : \mathcal{S} \to \mathbb{R}$ associates a weight (or cost) to each state $\mathbf{v}$. Let $\mathcal{V}' \subseteq \mathcal{V}$, the reduction $(\mathcal{S}', \mathcal{V}', \mathcal{C}') = \Pi_{\mathcal{V}'}(\mathcal{S}, \mathcal{V}, \mathcal{C})$ remains the same on states, and the new weight function $\mathcal{C}'$ is defined by:

$$\forall\, \mathbf{v}' \in \mathcal{S}', \quad \mathcal{C}'(\mathbf{v}') \;=\; \min_{\mathbf{v}\in\mathcal{S}\,:\,\mathbf{v}_{|\mathcal{V}'}=\mathbf{v}'} \mathcal{C}(\mathbf{v}) \tag{2.3}$$

When $\mathcal{V}' \not\subseteq \mathcal{V}$, we simply define $\Pi_{\mathcal{V}'}(\mathcal{S}, \mathcal{V}, \mathcal{C})$ as $\Pi_{\mathcal{V}'\cap\mathcal{V}}(\mathcal{S}, \mathcal{V}, \mathcal{C})$. For the composition, $(\mathcal{S}, \mathcal{V}, \mathcal{C}) = (\mathcal{S}_1, \mathcal{V}_1, \mathcal{C}_1) \wedge (\mathcal{S}_2, \mathcal{V}_2, \mathcal{C}_2)$ follows the same principle as above to compose states, with the extra rule

$$\mathcal{C}(\mathbf{v}_1 \wedge \mathbf{v}_2) \;=\; \mathcal{C}_1(\mathbf{v}_1) + \mathcal{C}_2(\mathbf{v}_2) \tag{2.4}$$

when $\mathbf{v}_1 \wedge \mathbf{v}_2$ is non-empty. And naturally $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$. We leave as an exercise the verification of (a3). The unit system is defined as $\mathbb{1} = (\{*\}, \emptyset, 0)$: it allows all states, and introduces no extra weight.

Systems with cost functions are closely related to Markov random fields, whence the name of "probabilistic systems:" Taking $\exp(-\mathcal{C})$, and renormalizing it by its sum over all states $\mathbf{v}$ in $\mathcal{S}$ yields a probability distribution on variables $\mathcal{V}$. For $(\mathcal{S},\mathcal{V},\mathcal{C}) = (\mathcal{S}_1,\mathcal{V}_1,\mathcal{C}_1) \wedge \ldots \wedge (\mathcal{S}_N,\mathcal{V}_N,\mathcal{C}_N)$, the cost function $\mathcal{C}_i$ represents the so-called *potential function* of *clique* $\mathcal{V}_i$, while the global cost function $\mathcal{C}$ is referred to as the *energy* function.

We have chosen the pair $(\min,+)$ to define reduction and composition. In the probabilistic interpretation above, reduction can thus be read as a maximum likelihood operation: the cost of a state corresponds to $-\log$ of its probability, so, in (2.3), likelihood is maximized over discarded variables. But other pairs than $(\min,+)$ would work as well, for example $(\max,+)$, $(\max,*)$ or $(+,*)$ (see [3]). For the latter, the reduction corresponds to a marginalization: one integrates the likelihood over the discarded variables.

Whatever the choice one makes, in practice cost functions are often handled under a renormalized form. This renormalization can be incorporated into the composition and reduction operators without altering their properties.

Language systems. This last example is borrowed to Rong Su's approach to distributed monitoring [102]. We define a language system $\mathcal{S} = (\mathcal{L},\mathcal{V})$ as a regular language $\mathcal{L}$ over a finite alphabet $\mathcal{V}$. So the letters in $\mathcal{V}$ define the variables on which this system operates, and we assume the existence of a maximal (finite) alphabet $\mathcal{V}_{max}$. The reduction $\Pi_{\mathcal{V}'}(\mathcal{S})$ is given by the natural projection of words in $\mathcal{L}$ on the sub-alphabet $\mathcal{V} \cap \mathcal{V}'$. And the composition $\mathcal{S}_1 \wedge \mathcal{S}_2$, with $\mathcal{S}_i = (\mathcal{L}_i,\mathcal{V}_i)$, is defined as

$$\mathcal{S}_1 \wedge \mathcal{S}_2 = (\mathcal{L},\mathcal{V}_1 \cup \mathcal{V}_2) \quad \text{with} \quad \mathcal{L} = \mathcal{L}_1 \times_L \mathcal{L}_1 \triangleq \Pi_{\mathcal{V}_1}^{-1}(\mathcal{L}_1) \cap \Pi_{\mathcal{V}_2}^{-1}(\mathcal{L}_2) \qquad (2.5)$$

where $\Pi_{\mathcal{V}_i}^{-1}$ denotes the reverse projection of $(\mathcal{V}_1 \cup \mathcal{V}_2)^*$ on $\mathcal{V}_i$. In other words, $\mathcal{L}$ is the usual parallel product of languages $\mathcal{L}_1$ and $\mathcal{L}_2$. It is a simple exercise to check axioms (a1) to (a4). Language systems are actually very close in nature to constraint systems. We'll see later that they enjoy the same properties.

In the next chapters, we will encode runs of distributed systems in a way similar to language systems. But instead of regular languages, we will have more elaborate data structures.

2.1.3 Graphs of a compound system

Hypergraph. As mentioned in the introduction, several graphical representations can be associated to a compound system $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$ operating on a set of variables. In the hypergraph representation, variables in $\mathcal{V}_{max}$ give the vertices, and each component $(\mathcal{S}_i,\mathcal{V}_i)$ defines the hyperedge $\mathcal{V}_i$ (Fig. 1.2.a). Without loss of generality, one can assume $\mathcal{V}_i \not\subseteq \mathcal{V}_j$ for $i \neq j$ (if inclusion happens, we replace $\mathcal{S}_i$ and $\mathcal{S}_j$ by the single component $\mathcal{S}_i \wedge \mathcal{S}_j$, operating on $\mathcal{V}_j$).

The interest of $\mathcal{H} = (\mathcal{V}_{max},\{\mathcal{V}_1,\ldots,\mathcal{V}_N\})$ is to display the interfaces between sets of components. Let $\mathcal{X},\mathcal{Y},\mathcal{Z} \subseteq \mathcal{V}_{max}$ be vertex sets, we say that $\mathcal{Y}$ separates $\mathcal{X}$ from $\mathcal{Z}$ (denoted $\mathcal{X}|\mathcal{Y}|\mathcal{Z}$) when on the hypergraph $\mathcal{H}_{|\mathcal{V}_{max}\setminus\mathcal{Y}}$, obtained by removing vertices $\mathcal{Y}$, no connected component contains vertices of both $\mathcal{X}$ and $\mathcal{Z}$. For

example, $\{V_1, V_2, V_3\}|\{V_4, V_7\}|\{V_7, V_8\}$ in Fig. 1.2.a. This allows us to say that variables $\{V_4, V_7\}$ capture all the interaction between components $\mathcal{S}_1$ and $\mathcal{S}_4$. And since $\{V_4, V_7\} \subseteq \mathcal{V}_2$, component $\mathcal{S}_2$ separates $\mathcal{S}_1$ from $\mathcal{S}_4$ as well, or is an interface between them.

This property is crucial to distributed algorithms. But before explaining why, we introduce a more convenient graphical representation to describe the interactions between components. This graph will form the support of our algorithms.

Connectivity graph, communication graph. The *connectivity graph* $\mathcal{G}^{cnx}$ of $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$ has $\{1, \ldots, N\}$ as vertices, or equivalently components $\mathcal{S}_i$, and (i, j) is an edge iff $\mathcal{V}_i \cap \mathcal{V}_j \neq \emptyset$ (Fig. 1.2.d).

A *communication graph* $\mathcal{G}^c$ for $\mathcal{S}$ is obtained by recursively removing redundant edges in the connectivity graph, until minimality is reached. An edge (i, j) is said to be *redundant* in a graph iff there exists a path $(i, k_1, k_2, \cdots, k_L, j)$ such that $\mathcal{V}_i \cap \mathcal{V}_j \subseteq \mathcal{V}_{k_l}$ and $k_l \notin \{i, j\}$ for $1 \leq l \leq L$. In other words, the direct interaction between $\mathcal{S}_i$ and $\mathcal{S}_j$ can be captured by the alternate path $(\mathcal{S}_i, \mathcal{S}_{k_1}, \cdots, \mathcal{S}_{k_L}, \mathcal{S}_j)$ of the graph.

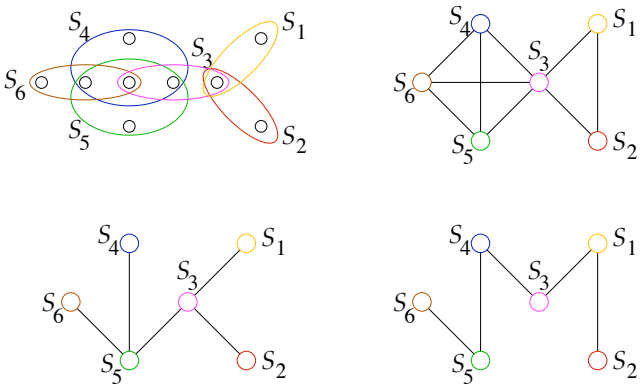


Figure 2.1: *The hypergraph $\mathcal{H}$ of a system with 6 components (top left), and the associated connectivity graph $\mathcal{G}^{cnx}$ of components (top right). Below, two communication graphs for this system.*

In general, a system has several communication graphs, as illustrated in figure 2.1. But we'll see that this is not really bothering. This is particularly true for the subclass of "tree shaped systems," that will play an important role in the sequel: We say that $\mathcal{S}$ *lives on a tree* iff one of its communication graphs is a tree. This class enjoys the following nice property:

Proposition 1 *If $\mathcal{S}$ lives on a tree, then* all *its communication graphs are trees.*

Separation property. Communication graphs of $\mathcal{S}$ are more helpful than hypergraphs to identify interfaces between sets of components. This is actually their *raison d'être*. Let us introduce some more notations. For an index set $I \subseteq \{1, \ldots, N\}$, we define $\mathcal{S}_I \triangleq \wedge_{i \in I} \mathcal{S}_i$ and $\mathcal{V}_I \triangleq \cup_{i \in I} \mathcal{V}_i$. Let $I, J, K \subseteq \{1, \ldots, N\}$ be index sets, and

consider the aggregated components $\mathcal{S}_I, \mathcal{S}_J, \mathcal{S}_K$. We say that $\mathcal{S}_J$ *separates* $\mathcal{S}_I$ from $\mathcal{S}_K$ in $\mathcal{S}$ iff $\mathcal{V}_I|\mathcal{V}_J|\mathcal{V}_K$ on $\mathcal{H}$.

Proposition 2 *If $I|J|K$ on a communication graph $\mathcal{G}^c$ of $\mathcal{S}$, then $\mathcal{S}_J$ separates $\mathcal{S}_I$ from $\mathcal{S}_K$ in $\mathcal{S}$.*

The separation property read on $\mathcal{G}^c$ is actually a fast way of identifying (some of the) cases where axiom (a3) applies, and forms the basis of message passing algorithms, as we show in the next section.

2.2 Distributed reduction algorithms

2.2.1 The reduction Problem

Composition is a natural tool to build large complex systems from small simple components. In many applications, the large system $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$ becomes intractable. Fortunately, one is generally not so much interested in "computing" the large system $\mathcal{S}$, but rather in understanding its influence on a given component $\mathcal{S}_i$. Specifically, once inserted into $\mathcal{S}$, a component $\mathcal{S}_i$ changes and becomes $\mathcal{S}_i' \triangleq \Pi_{\mathcal{V}_i}(\mathcal{S})$. Computing these $\mathcal{S}_i'$ defines what we call the *reduction problem*. Naturally, one would like to determine or approximate these reduced components *without computing $\mathcal{S}$ itself*. Our objective is thus to obtain the $\mathcal{S}_i'$ with *local* computations, *i.e.* computations performed at the scale of a component, and to highlight some of their properties.

Before, and to illustrate the scope of this approach, we give an application example of the reduction problem in coding theory, which is much different from the problems we shall consider in the next chapters.

The example concerns the decoding of the so-called low-density parity check (LDPC) codes. These error correcting codes are constructed in the following way : codewords have a length of N bits, represented as variables $B_1, \ldots, B_N$ taking values 0 or 1. The code is obtained by forbidding some configurations among the 2^N possible ones. Specifically, M (independent) linear constraints $\mathcal{S}_1, \ldots, \mathcal{S}_M$ are applied to these variables. Each $\mathcal{S}_i$ involves a *small* subset of bits $\mathcal{V}_i \subseteq \{B_1, \ldots, B_N\}$ and allows states satisfying $\sum_{B_n \in \mathcal{V}_i} B_n = 0$, where addition is modulo 2. The number of possible values for $\mathbf{b} = (b_1, \ldots, b_N)$ thus reduces from 2^N to 2^K, with $K = N - M$, which corresponds to a rate $\frac{K}{N}$ code.

Case 1. Let us consider first the decoding problem when an LDPC code is used over an erasure channel. This random channel erases a transmitted bit with probability p, and transmits it perfectly with probability $1 - p$. The decoding problem consists in recovering the transmitted codeword $\mathbf{b} = (b_1, \ldots, b_N)$ from received values $\mathbf{r} = (r_1, \ldots, r_N)$, where r_n is either 0, 1 or x, standing for "erased." This takes the form of a big linear system, one equation per constraint, where B_n is set to r_n if 0 or 1 was received, and left as an unknown otherwise.

In our setting, for each observation r_n let us build a system $\mathcal{R}_n$ operating on variable B_n and pinning its value to r_n if 0 or 1 was received, or allowing both

values otherwise. The global decoding means computing $\mathcal{S} \wedge \mathcal{R}_1 \wedge \ldots \wedge \mathcal{R}_N$, made of codewords that match observations $\mathbf{r} = (r_1, \ldots, r_N)$. There is no obvious way to perform this global computation efficiently. Moreover, it can result in a huge set if $\mathbf{r}$ is not uniquely decodable. One would rather prefer to identify the value of bits B_n that can be recovered, and leave the others as "unknown." Possibly with computations involving only a few bits at a time. The decoding of each bit B_n is given by $\Pi_{B_n}(\mathcal{S} \wedge \mathcal{R}_1 \wedge \ldots \wedge \mathcal{R}_N)$, which is a sub-product of the $\Pi_{\mathcal{V}_i}(\mathcal{S} \wedge \mathcal{R}_1 \wedge \ldots \wedge \mathcal{R}_N)$. If $\mathbf{r} = (r_1, \ldots, r_N)$ is decodable, this projection assigns a single value to each B_n. Otherwise, some undecodable bits remain, that can still take both values.

Case 2. Let us consider now the transmission of an LDPC code over, say, a Gaussian channel. B_n is modulated as $+1$ or -1 and corrupted by the additive Gaussian noise Z_n, which yields observation $R_n = (2 * B_n - 1) + Z_n$ taking values in $\mathbb{R}$. We now model systems $\mathcal{S}_i$ as systems with a weight function : they allow the same local states as above, and assign a null weight to each of them. This stands for the equiprobability of all codewords, weights being homogeneous to a log likelihood. We build observation systems as follows : given the received value r_n, system $\mathcal{R}_n$ operates on B_n and assigns weights $\log \mathbb{P}(r_n|B_n = 0), \log \mathbb{P}(r_n|B_n = 1)$ to values 0 and 1 of B_n. Then, in the $(\max, +)$ setting, the reduced system $\Pi_{B_n}(\mathcal{S} \wedge \mathcal{R}_1 \wedge \ldots \wedge \mathcal{R}_N)$ allows values 0 and 1 to B_n with weights

$$\log \max_{b_i,\, 1 \leq i \leq N,\, i \neq n} \mathbb{P}(b_1, \ldots, b_{n-1}, 0/1, b_{n+1}, \ldots, b_N | r_1, \ldots, r_N) + C \tag{2.6}$$

where C is a constant. Therefore, these values allow a maximum likelihood decoding of bit B_n. This statement may be more convincing without the log, *i.e.* in a $(\max, *)$ setting. Let us assign weight 1 to configurations of $\mathcal{S}_i$, still for equiprobability (any constant value would work as well). Observation systems now assign $\mathbb{P}(r_n|B_n = 0/1)$ to values 0 and 1 of B_n. Then $\Pi_{B_n}(\mathcal{S} \wedge \mathcal{R}_1 \wedge \ldots \wedge \mathcal{R}_N)$ yields weights proportional to

$$\max_{b_i,\, 1 \leq i \leq N,\, i \neq n} \mathbb{P}(b_1, \ldots, b_{n-1}, 0/1, b_{n+1}, \ldots, b_N, r_1, \ldots, r_N) \tag{2.7}$$

2.2.2 Message passing algorithm

The message passing algorithm (MPA) solves the reduction problem relying on the separation criterion between components. The latter induces the following two computation rules.

Consequences of the separation criterion. Let I, J, K be pairwise distinct index sets, and assume that $\mathcal{S}_J$ separates $\mathcal{S}_I$ from $\mathcal{S}_K$ in $\mathcal{S}$, then :

$$\Pi_{\mathcal{V}_J}(\mathcal{S}_{I \cup J \cup K}) \quad = \quad \Pi_{\mathcal{V}_J}(\mathcal{S}_I) \wedge \mathcal{S}_J \wedge \Pi_{\mathcal{V}_J}(\mathcal{S}_K) \tag{2.8}$$

(2.8) is known as a *merge* equation. It expresses that if a system $\mathcal{S}_J$ separates two (or more) components, the latter have independent influences on $\mathcal{S}_J$. The proof

mostly uses (a3) :

$$\begin{aligned}\Pi_{\mathcal{V}_J}(\mathcal{S}_{I\cup J\cup K}) &= \Pi_{\mathcal{V}_J}(\mathcal{S}_I \wedge \mathcal{S}_J \wedge \mathcal{S}_K) \\ &= \Pi_{\mathcal{V}_J}(\mathcal{S}_J) \wedge \Pi_{\mathcal{V}_J}(\mathcal{S}_I \wedge \mathcal{S}_K) \\ &= \mathcal{S}_J \wedge \Pi_{\mathcal{V}_J}(\mathcal{S}_I) \wedge \Pi_{\mathcal{V}_J}(\mathcal{S}_K) \end{aligned} \tag{2.9}$$

The second consequence of separation expresses that the influence of $\mathcal{S}_K$ on $\mathcal{S}_I$ can be propagated through the intermediate system $\mathcal{S}_J$, which is known as the *propagation* equation :

$$\Pi_{\mathcal{V}_I}(\mathcal{S}_{I\cup J\cup K}) = \mathcal{S}_I \wedge \Pi_{\mathcal{V}_I}[\mathcal{S}_J \wedge \Pi_{\mathcal{V}_J}(\mathcal{S}_K)] \tag{2.10}$$

The proof uses both (a3) and (a1) :

$$\begin{aligned}\Pi_{\mathcal{V}_I}(\mathcal{S}_{I\cup J\cup K}) &= \mathcal{S}_I \wedge \Pi_{\mathcal{V}_I}(\mathcal{S}_J \wedge \mathcal{S}_K) \\ &= \mathcal{S}_I \wedge \Pi_{\mathcal{V}_I}[\Pi_{\mathcal{V}_J\cup\mathcal{V}_K}(\mathcal{S}_J \wedge \mathcal{S}_K)] \\ &= \mathcal{S}_I \wedge \Pi_{\mathcal{V}_I}[\Pi_{\mathcal{V}_J}(\mathcal{S}_J \wedge \mathcal{S}_K)] \\ &= \mathcal{S}_I \wedge \Pi_{\mathcal{V}_I}[\mathcal{S}_J \wedge \Pi_{\mathcal{V}_J}(\mathcal{S}_K)] \end{aligned} \tag{2.11}$$

The key is that $\Pi_{\mathcal{V}_I} \circ \Pi_{\mathcal{V}_J\cup\mathcal{V}_K} = \Pi_{\mathcal{V}_I} \circ \Pi_{\mathcal{V}_J}$, due to the separation property. Of course, by (a1) and taking into account that $\mathcal{S}_I$ operates on $\mathcal{V}_I$, a term like $\Pi_{\mathcal{V}_J}(\mathcal{S}_I)$ can be replaced by $\Pi_{\mathcal{V}_I\cap\mathcal{V}_J}(\mathcal{S}_I)$.

Distributed reduction algorithm. Assume $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$ lives on a tree, and $\mathcal{G}^c$ is one of its communication graphs. Let $\mathcal{N}(i)$ denote the neighbors of vertex i on $\mathcal{G}^c$, $1 \leq i \leq N$. The reduction algorithm for $\mathcal{S}$ is based on messages exchanged between neighbors : each system $\mathcal{S}_i$ maintains and updates a message $\mathcal{M}_{i,j}$ for each neighbor $\mathcal{S}_j$, so there are two messages per edge (i,j) of $\mathcal{G}^c$, one in each direction. The idea is that $\mathcal{M}_{i,j}$ collects information about $\mathcal{S}_j$ in systems located on the side of $\mathcal{S}_i$ with respect to edge (i,j), relying on the fact that system $\mathcal{S}_i$ separates $\mathcal{S}_j$ from the branches behind $\mathcal{S}_i$. The set of systems contributing to the message grows until the whole branch beyond i has been covered.

Algorithm $\mathbf{A}_1$

1. Initialization

$$\mathcal{M}_{i,j} = \mathbb{1}, \qquad \forall (i,j) \in \mathcal{G}^c \tag{2.12}$$

2. Until stability of messages, select an edge (i,j) and apply the update rule

$$\mathcal{M}_{i,j} := \Pi_{\mathcal{V}_i\cap\mathcal{V}_j}[\mathcal{S}_i \wedge (\bigwedge_{k\in\mathcal{N}(i)\setminus j} \mathcal{M}_{k,i})] \tag{2.13}$$

3. Termination

$$\mathcal{S}'_i = \mathcal{S}_i \wedge (\bigwedge_{k\in\mathcal{N}(i)} \mathcal{M}_{k,i}), \qquad 1 \leq i \leq N \tag{2.14}$$

The termination equation (2.14) is obviously a merge, while the update equation (2.13) mixes a merge and a propagation (Fig. 2.2). Observe that (2.13) merges incoming messages of all edges around i excepted the edge (i, j) on which a new message will be sent.

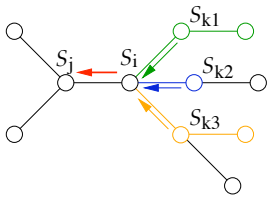


Figure 2.2: *Messages arriving at $\mathcal{S}_i$ gather information of their sub-tree, and are combined to form a message to $\mathcal{S}_j$.*

Convergence.

Theorem 1 *Let $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$ live on a tree, and $\mathcal{G}^c$ be a communication graph for $\mathcal{S}$. Then $\mathbf{A}_1$ converges in finitely many steps, and at convergence $\mathcal{S}'_i = \Pi_{\mathcal{V}_i}(\mathcal{S})$, $1 \leq i \leq N$.*

"Steps" refer to the number of message updates, where only updates changing the value of a message are counted. Notice that the *scheduling* of the algorithm, *i.e.* the choice of the edge (i, j) at each step, is left unspecified. This property will thus lead to asynchronous distributed algorithms in the sequel.

Evolving systems. Surprisingly, this result can be easily extended to *evolving systems,* which will be useful in the next chapters. Assume components $\mathcal{S}_i$ are not fixed once for all, but may evolve in time, which we denote by $\mathcal{S}_i(t)$, $t \in \mathbb{N}$. We make no other assumption on this evolution than

$$\exists T_i < \infty \,:\, \forall t > T_i,\ \mathcal{S}_i(t) = \mathcal{S}_i(T_i) \tag{2.15}$$

i.e. the components stabilize. Assume Algorithm 1 is started at time $t = 0$, and that each step of the recursion takes one unit of time. So components change after each message update. Then theorem 1 above still holds, with $\mathcal{S}$ replaced by the stabilized system $\mathcal{S}_1(T_1) \wedge \ldots \wedge \mathcal{S}_N(T_N)$.

Of course, in the transient part of the algorithm, the messages combined in (2.13) are desynchronized, *i.e.* they gather information collected in different components at different times. This is the price to pay to get an asynchronous algorithm. One can probably refine this result with extra assumptions. For example, with a monotonic evolution of components and specific constraints on the scheduling of the algorithm[2], there certainly exists some form of monotony on messages. We leave this for future research.

[2]the ordering in which messages are updated

2.2.3 Turbo algorithms

In practical applications, few systems have a tree structure. To solve the reduction problem in more general cases, one may adopt several strategies.

The simplest one consists in aggregating some components into macro-components, in such a way that these macro-components interact according to a tree structure. There exists a systematic way to do that, by first triangulating a communication graph of $\mathcal{S}$, and then taking cliques of nodes to form the macro-components. It can indeed be shown that the cliques of a triangulated graph have a tree shaped interaction structure. This is also called the "junction tree technique." The price to pay is that computations of the MPA are then performed on larger components, which generally means a loss in efficiency. And finding the best triangulated graph, or equivalently the tree with the smallest macro-components, has been shown to be NP hard. Moreover, even with reasonably dense communication graph, the number of aggregated components rapidly brings us back to an intractable reduction problem. In the case of a grid of components, for example, the typical aggregation would gather all components of a row (or of a column), and organize them in a chain.

The less known *conditioning method* does a similar thing. It freezes one (or several) variable(s) to a particular value, which amounts to removing it (them) from $\mathcal{S}$, and may thus open a cycle. The reduction problem can be solved easily on the remaining "conditioned system," if it lives on a tree. The complexity is similar to the junction tree method : all combinations of values must be explored for the conditioning variables. And a technical difficulty remains in the deconditioning step, which combines all reduced components obtained for all values of the frozen variables.

In practice, the complexity of exact reduction methods explodes with the number of cycles in the communication graph of the system. This doesn't mean however that we must stop here our quest for methods to handle large distributed systems. In the digital communication community, people have discovered that running the MPA on graphs with cycles could sometimes provide excellent results [10], even if this is theoretically illegal ! The MPA are then called *turbo algorithms,* because the message sent on a branch may be propagated along a cycle of the graph and eventually come back to its transmitter after some transformations. Just like the power of exhaust gases of an engine drives the compression of fresh air at the admission side. We now examine some of their properties.

Conditions for convergence. To study the convergence of message passing algorithms on graphs with cycles, we introduce a weak notion of "topology" on systems, with extra axioms defining its relations with $\wedge$ and Π. Let us assume the existence of a partial order $\Subset$ on systems, where $\mathcal{S} \Subset \mathcal{S}'$ can be read as $\mathcal{S}$ *contains more information than* $\mathcal{S}'$. We require $\Subset$ to satisfy the following properties :

$$\forall \mathcal{S}, \quad \mathcal{S} \Subset \mathbb{1} \tag{a5}$$

which means that $\mathbb{1}$ is the least informative system.

$$\forall \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \qquad \mathcal{S}_1 \Subset \mathcal{S}_2 \quad \Rightarrow \quad \mathcal{S}_1 \wedge \mathcal{S}_3 \Subset \mathcal{S}_2 \wedge \mathcal{S}_3 \tag{a6}$$

$$\forall \mathcal{S}, \mathcal{S}', \forall \mathcal{V} \subseteq \mathcal{V}_{max}, \qquad \mathcal{S} \Subset \mathcal{S}' \quad \Rightarrow \quad \Pi_{\mathcal{V}}(\mathcal{S}) \Subset \Pi_{\mathcal{V}}(\mathcal{S}') \tag{a7}$$

Intuitively, (a6) states that composition incorporates the same "amount" of information to all systems, and (a7) means that reduction can't introduce information.

As an example of this situation, consider constraint systems: $\mathcal{S} \Subset \mathcal{S}'$ can be simply defined by $Span(\mathcal{S}) \subseteq Span(\mathcal{S}')$, which entails in particular $\mathcal{V}_{\mathcal{S}} \supseteq \mathcal{V}_{\mathcal{S}'}$, *i.e.* $\mathcal{S}$ constrains the value of more variables.

Under these conditions, one easily shows that the messages $\mathcal{M}_{i,j}$ in (2.13) have a decreasing evolution for $\Subset$, regardless of the structure of the communication graph. In other words, information augments at each node with the exchange of messages. Moreover

Theorem 2 *Under axioms (a5,a6,a7), algorithm* $\mathbf{A}_1$ *has at most one accessible stationary point.*

Notice that the update equation (2.13) may admit several stationary sets of messages $\mathcal{M}_{i,j}$, but at most one of them is accessible from the initial value of $\mathbf{A}_1$. The accessibility refers to a denumerable number of steps[3]. If the number of possible states is bounded, (*i.e.* $|\mathcal{D}_{\mathcal{V}_{max}}| < \infty$), the stationary point is reached in a finite number of steps, whatever the ordering of updates. For infinite state spaces, one may be able to refine this result and show that convergence is actually granted for a weak topology. This is the case for language systems for example (section 2.1.2), assuming the usual notion of weak convergence for non commutative formal series.

It would be nice if this simple approach could hold for random systems. Unfortunately, this is not the case: simple attempts at defining $\Subset$ for systems with cost functions fail, even if no renormalization operation is introduced on cost functions. There is little hope of success since some authors have built examples of systems for which the MPA does not converge to a fix point [79]. Nevertheless, convergence properties of MPAs on random systems are now quite well understood [104, 58, 92, 93, 94, 20], and there exist tools to test it *a priori*. Convergence deeply relies on the sparseness of the communication graph, or in other words, on the length of cycles in the graph. Roughly speaking, cycles must be long enough to introduce a decorrelation between an outgoing message and its version coming back after a cyclic propagation. This ensures that the messages merged at a node are almost independent, as they are on a tree.

To summarize, one can consider systems with cost functions as having a double nature. First, components in $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$ carry hard constraints, for which convergence of the MPA is guaranteed, whatever the graph of $\mathcal{S}$ is. But components also carry soft constraints, defined by the likelihood of the remaining states, after hard constraints have operated their selection. For these soft constraints, convergence must be studied with the standard tools dedicated to turbo algorithms.

Properties at convergence. Let us now consider properties of stationary points of $\mathbf{A}_1$, assuming there exist some.

[3]We only count message updates that change the content of a message.

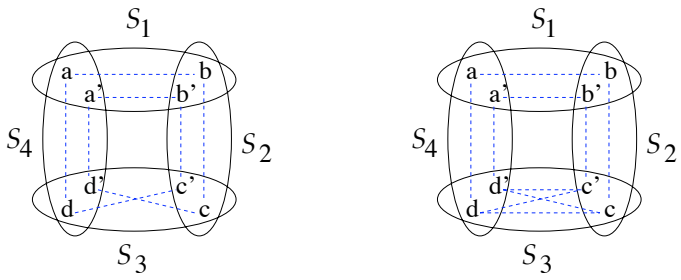


Figure 2.3: *Two examples of constraint systems defined on variables* $\{A, B, C, D\}$. *Interpretation example for these graphics : component* $\mathcal{S}_1$ *operates on* $\{A, B\}$ *and contains states* (a, b) *and* (a', b').

Our aim was to obtain the reduced components $\Pi_{\mathcal{V}_i}(\mathcal{S})$. Let us first remark that the $\mathcal{S}'_i$ obtained by (2.14) at a stationary point of (2.13) generally differ from the true $\Pi_{\mathcal{V}_i}(\mathcal{S})$, when the global system $\mathcal{S}$ doesn't live on a tree, of course. Fig. 2.3 (left) gives a counter-example, in the case of constraint systems : $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_4$ is empty, so the true reduced components $\Pi_{\mathcal{V}_i}(\mathcal{S})$ are also empty. But the $\mathcal{S}'_i$ obtained at convergence of algorithm $\mathbf{A}_1$ are identical to the $\mathcal{S}_i$. In fact, the situation is even worse : in some cases the true reduced components can't be obtained as a stationary point of $\mathbf{A}_1$. Fig. 2.3 (right) illustrates this case : $\Pi_{\mathcal{V}_3}(\mathcal{S})$ contains states (c, d) and (c', d'), and otherwise $\Pi_{\mathcal{V}_i}(\mathcal{S}) = \mathcal{S}_i$ for $i \neq 3$. But there exists no set of stationary messages $\mathcal{M}_{i,j}$ that would yield these reduced components by (2.14). Nevertheless, and despite these drawbacks, the $\mathcal{S}'_i$ computed by $\mathbf{A}_1$ are far from being meaningless, as we show now and in the next section.

The *local extendibility* is probably the most striking property of the $\mathcal{S}'_i$ obtained at a stationary point of (2.13).

Theorem 3 *Let* $\mathcal{G}^c$ *be a communication graph of* $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$, *let the* $\mathcal{M}_{i,j}$ *be a stationary point of (2.13) on* $\mathcal{G}^c$ *and let the* $\mathcal{S}'_i$ *be derived by (2.14). Select* $J \subseteq \{1, 2, \ldots, N\}$ *such that the subgraph* $\mathcal{G}^c{}_{|J}$ *is a tree, and define* $\bar{\mathcal{S}}_i = \mathcal{S}_i \wedge (\bigwedge_{k \in \mathcal{N}(i) \setminus J} \mathcal{M}_{k,i})$, $1 \leq i \leq N$. *Then* $\forall j \in J$, $\mathcal{S}'_j = \Pi_{\mathcal{V}_j}(\bar{\mathcal{S}}_J)$.

The proof is intuitively simple : replacing the $\mathcal{S}_i$ by $\bar{\mathcal{S}}_i$ and running $\mathbf{A}_1$ yields the same messages between the remaining nodes. Since the latter form a tree, the result follows by theorem 1.

To clarify the interest of theorem 3, Let us take the example of constraint systems. Consider a local state $\mathbf{v}_i$ in $\mathcal{S}'_i$. Since $\mathbf{A}_1$ is an approximation, $\mathbf{v}_i$ is not necessarily the projection of a global state $\mathbf{v}$ of $\mathcal{S}$. Nevertheless, $\mathbf{v}_i$ can be extended into a larger state $\mathbf{v}_J$ over *any tree* around i ($\mathbf{v}_J$ may not involve all variables of $\mathcal{S}$, however). So only a cycle of $\mathcal{G}^c$ could determine that a $\mathbf{v}_i$ in some system $\mathcal{S}'_i$ doesn't belong to the true $\Pi_{\mathcal{V}_i}(\mathcal{S})$ (see the example of fig. 2.3). In other words, $\mathbf{A}_1$ is blind to cycles. In the case of language systems, the same interpretation holds for words in the reduced language $\mathcal{S}'_i$. The case of systems with cost functions is examined in more details below.

The result may look weak since J could remain quite small and involve few components, and thus few variables of $\mathcal{V}_{max}$ (see Fig. 2.4, left). In reality, in many

settings it is possible to duplicate components and relate them by an equality constraint, in order to introduce fake new components (see Fig. 2.4, right). A stationary point of $\mathbf{A}_1$ easily extends to a stationary point on this expanded system, for which the J set now reaches all components of the original system. As a result, a $\mathbf{v}_i$ can be extended to a $\mathbf{v}_J$ covering all components of the original system $\mathcal{S}$, but "extremities don't match," *i.e.* the values taken by $\mathbf{v}_J$ in the various copies of a component may differ.

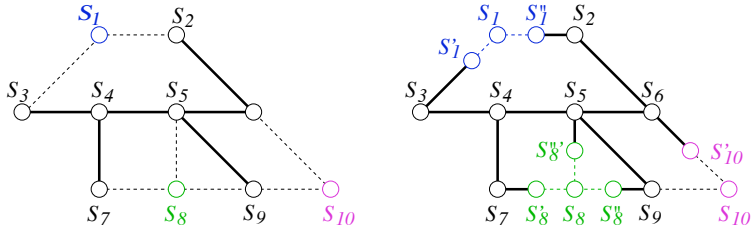


Figure 2.4: *Left: Communication graph relating 10 components, and a nested tree defined by* $J = \{2, 4, 5, 6, 7, 9\}$ *(solid edges). Right: by duplicating components* $\mathcal{S}_1, \mathcal{S}_8$ *and* $\mathcal{S}_{10}$*, and relating copies by an equality constraint, one can actually define a nested tree reaching all components (possibly several times for the duplicated ones).*

Moving to probabilistic systems, the local extendibility can be interpreted as a *local-tree optimality* property, already mentioned in [108]. Let us consider positive weight functions in the $(max, *)$ setting, and assume systems are normalized: $\max_{\mathbf{v}\in\mathcal{S}} \mathcal{C}(\mathbf{v}) = 1$. This requires that the composition $\wedge$ contain a normalizing operation (normalization is already preserved by reductions).

When $\mathcal{S}$ lives on a tree, $\mathbf{A}_1$ converges and yields the $\mathcal{S}'_i = \Pi_{\mathcal{V}_i}(\mathcal{S})$. It actually solves a dynamic programming problem. Let $\mathbf{v}^\star$ (resp. $\mathbf{v}_i^\star$) denote a state of $\mathcal{S}$ (resp. $\mathcal{S}'_i$) having weight 1, *i.e.* a most likely state of $\mathcal{S}$ (resp. $\mathcal{S}'_i$) in the probabilistic interpretation of weights. Clearly, the restriction of a $\mathbf{v}^\star$ to variables $\mathcal{V}_i$ necessarily yields a $\mathbf{v}_i^\star$, and conversely a $\mathbf{v}_i^\star$ is necessarily part of at least one $\mathbf{v}^\star$. So, if components $\mathcal{S}'_i$ contain a single $\mathbf{v}_i^\star$ at convergence of $\mathbf{A}_1$, these local states can be composed to form the *unique* optimal state $\mathbf{v}^\star$ of $\mathcal{S}$: $\{\mathbf{v}^\star\} = \bigwedge_i \{\mathbf{v}_i^\star\}$. This is a well known property in dynamic programming.

In the case where $\mathcal{S}$ doesn't live on a tree, assume that some local optima $\mathbf{v}_i^\star$ can be assembled into a valid global state $\mathbf{v}^\star$ of $\mathcal{S}$ (in terms of hard constraints). There is *a priori* no reason that $\mathcal{C}(\mathbf{v}^\star) = 1$, so how good is this $\mathbf{v}^\star$ in terms of cost function? Let J be an index set such that $\mathcal{G}^c{}_{|J}$ is a tree, let I be its complement in $\{1, \ldots, N\}$, and let us introduce variable sets $\mathcal{V}'_I = \mathcal{V}_I \setminus \mathcal{V}_J$, $\mathcal{V}_{I,J} = \mathcal{V}_I \cap \mathcal{V}_J$ and $\mathcal{V}'_J = \mathcal{V}_J \setminus \mathcal{V}_I$. Then

$$\forall \mathbf{v}'_J, \quad [\ (\mathbf{v}'_J, \mathbf{v}^\star_{I,J}, \mathbf{v}'^\star_I) \in \mathcal{S} \quad \Rightarrow \quad \mathcal{C}(\mathbf{v}'_J, \mathbf{v}^\star_{I,J}, \mathbf{v}'^\star_I) \leq \mathcal{C}(\mathbf{v}'^\star_J, \mathbf{v}^\star_{I,J}, \mathbf{v}'^\star_I)\ ] \quad (2.16)$$

so changing the value of $\mathbf{v}_i^\star$ on any tree around node i will not improve the cost function to optimize. The MPA (also called belief propagation in this case) thus converges towards a local optimum of the cost function of $\mathcal{S}$, but its basin of attraction is reasonably large.

2.2.4 Involutive systems

Some families of systems enjoy a useful property that we call *involutivity*. This property states that systems do not change when composed with part of themselves :

$$\forall \mathcal{S}, \forall \mathcal{V}, \;\; \mathcal{S} \wedge \Pi_{\mathcal{V}}(\mathcal{S}) \;=\; \mathcal{S} \tag{a8}$$

(We say that $\Pi_{\mathcal{V}}(\mathcal{S})$ is *absorbed by* $\mathcal{S}$.) Observe that $\Pi_{\mathcal{V}}(\mathbb{1}) = \mathbb{1}$ comes as an immediate consequence of (a8).

Involutivity is a strong property. It is clearly satisfied by constraint systems, by language systems, and in general corresponds to systems defined by some form of constraint set. But systems with cost functions are not involutive.

Proposition 3 *Assuming (a8), let $\mathcal{S}_i$ operate on $\mathcal{V}_i$, $1 \leq i \leq N$, then*

$$\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N \quad \Rightarrow \quad \mathcal{S} = \Pi_{\mathcal{V}_1}(\mathcal{S}) \wedge \ldots \wedge \Pi_{\mathcal{V}_N}(\mathcal{S}) \tag{2.17}$$

The proof essentially uses (a3). In words, if $\mathcal{S}$ *factorizes*, the *reduced* components $\mathcal{S}'_i \triangleq \Pi_{\mathcal{V}_i}(\mathcal{S})$ give another factorization of $\mathcal{S}$, named the *canonical factorization*. In the next chapters, we will also use this property to build a *minimal product covering* of a system $\mathcal{S}$.

Involutivity has many other nice consequences. For example on the convergence of $\mathbf{A}_1$: let us define $\Subset$ by

$$\mathcal{S}_1 \Subset \mathcal{S}_2 \quad \Leftrightarrow \quad \mathcal{S}_1 \wedge \mathcal{S}_2 = \mathcal{S}_1 \tag{2.18}$$

(*i.e.* $\mathcal{S}_1$ absorbs $\mathcal{S}_2$). Then

Proposition 4 $\Subset$ *is a partial order relation that satisfies axioms (a5,a6,a7).*

So theorem 2 holds. This means that messages collect more and more information in algorithm $\mathbf{A}_1$. Moreover, $\mathbf{A}_1$ actually performs a progressive reduction of components $\mathcal{S}_i$:

Theorem 4 *Define the $\mathcal{S}'_i$ by (2.14) at any step of $\mathbf{A}_1$. In an involutive setting, one has $\mathcal{S} = \mathcal{S}'_1 \wedge \ldots \wedge \mathcal{S}'_N$ at any time in $\mathbf{A}_1$. Moreover, the $\mathcal{S}'_i$ computed at each step form a decreasing sequence for $\Subset$, and they satisfy $\Pi_{\mathcal{V}_i}(\mathcal{S}) \Subset \mathcal{S}'_i \Subset \mathcal{S}_i$.*

As we have already mentioned, this progressive reduction may stop at (or converge to) a stationary point located "above" the minimal term $\Pi_{\mathcal{V}_i}(\mathcal{S})$. With theorem 2 in mind, this means that only "cycles of constraints" on $\mathcal{G}^c$ would allow to go further in the reduction, as illustrated by the examples in Fig. 2.3.

Philosophically, involutive systems are defined through constraints. And since adding redundant constraints is harmless, one gets several extra properties. For example, an alternate reduction algorithm can be derived.

Algorithm $\mathbf{A}_2$

1. Initialization

$$\tilde{\mathcal{S}}'_i \;=\; \mathit{1\!\!1}, \qquad 1 \leq i \leq N \tag{2.19}$$

2. Until stability of reduced systems, select a vertex i and apply the update rule

$$\tilde{\mathcal{S}}'_i \;:=\; \mathcal{S}_i \wedge [\bigwedge_{k \in \mathcal{N}(i)} \Pi_{\mathcal{V}_i \cap \mathcal{V}_k}(\tilde{\mathcal{S}}'_k)] \tag{2.20}$$

This algorithm derives directly from $\mathbf{A}_1$: let us replace (2.13) by a more symmetric expression that would use *all* incoming messages to compute an outgoing message :

$$\tilde{\mathcal{M}}_{i,j} \;:=\; \Pi_{\mathcal{V}_i \cap \mathcal{V}_j}[\mathcal{S}_i \wedge (\bigwedge_{k \in \mathcal{N}(i)} \tilde{\mathcal{M}}_{k,i})] \tag{2.21}$$

In other words, one sends back to node $\mathcal{S}_j$ its own message $\tilde{\mathcal{M}}_{j,i}$ to $\mathcal{S}_i$. In an involutive setting, this will have no impact on $\mathcal{S}_j$ so the modified algorithm will behave like $\mathbf{A}_1$. At this point, observe that messages become useless : computations can be equivalently described in terms of systems $\mathcal{S}'_i$ as in $\mathbf{A}_2$. The latter has the same shape of a Gauss-Seidel procedure to solve linear systems of equations. Not surprisingly, one has

Theorem 5 *In an involutive setting, let $\mathcal{G}^c$ be a communication graph for $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$. Algorithms $\mathbf{A}_1$ and $\mathbf{A}_2$ have the same accessible stationary point on $\mathcal{G}_c$ (in terms of components $\mathcal{S}'_i$ and $\tilde{\mathcal{S}}'_i$), if this point exists for one of them.*

Recall that there exists at most one such stationary point, by theorem 2. In particular, if $\mathcal{S}$ lives on a tree, algorithm $\mathbf{A}_2$ converges in finite time to the exact reduced components.

One can go further in terms of using redundant constraints : instead of incorporating extra information in the update equation, let us now reincorporate redundant edges in the communication graph of $\mathcal{S}$:

Theorem 6 *In an involutive setting, let $\mathcal{S} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_N$, and $\mathcal{G}^{cnx}$ be the connectivity graph of $\mathcal{S}$. Let graph $\mathcal{G} \subseteq \mathcal{G}^{cnx}$ be obtained by removing some redundant edges in $\mathcal{G}^{cnx}$. Whatever the choice of $\mathcal{G}$, $\mathbf{A}_1$ (resp. $\mathbf{A}_2$) yields the same result in terms of components $\mathcal{S}'_i$ (resp. $\tilde{\mathcal{S}}'_i$).*

These results illustrate the power of (a8). Many other properties can be derived for involutive systems, in particular in terms of local extendibility. We refer the reader to [38] for details.

2.3 Summary

We have introduced an abstract notion of system, operating on a reduced set of variables, and provided with a composition operation. This allows us the construction of large systems from components. The interactions of components can be described as a connectivity graph, or more specifically as a communication graph.

We have also introduced the concept of reduction of a large system to part of its variables. Since large systems can't be studied globally, this allows us to study them by parts. In particular, for a compound system, the reduction explains how a component's behavior is modified once the latter is connected to all the others. Computing reduced components of a large system is a powerful key to answer many problems related to distributed systems.

As soon as composition and projection satisfy a small set of axioms, one can derive efficient algorithms to compute reduced components. These message passing algorithms are by nature distributed (or distributable) and based on asynchronous computations. They converge in finite time to the reduced components for systems whose communication graphs are trees, whatever the ordering of computations. On general graphs, they converge when systems are defined in terms of constraints, but may provide only approximate reduced components. When systems also incorporate a notion of cost function, reduction algorithms "generally" converge if the communication graph is sparse enough, as revealed by the results on turbo-algorithms. Although the convergence point is only an approximation of the desired reduced components, several properties of the latter suggest a reasonable quality of this approximation.

The next chapters rely on this formalism to develop distributed algorithms for networks of dynamic systems. So the essential changes appear in the details of system definitions : they become dynamic systems, which introduces the notion of time in this setting.

Related work

The idea of message passing algorithms has appeared in several communities, under different names and sometimes independently. It is known as dynamic programming in computer science, as the Kruskal or Dijkstra algorithm in graph theory, as the Viterbi algorithm in digital communications, etc. So let us simply mention some references in statistics [76, 77, 87] : This community specifically studied the topic, with a focus on inference problems, of course, but also on different formalisms to describe statistical dependencies (various graphical models, semi-matroids, etc.). A large effort is dedicated to learning problems, to discover the underlying graphs and possibly causality relations.

Surprisingly, the generalization of message passing algorithms into turbo algorithms occurred elsewhere, and for casual reasons. The discovery was triggered by the spectacular properties of turbo-codes [10], and it took (little) time to establish the connection between turbo-algorithms and MPA [80, 69, 70]. Convergence properties of this approximate procedure have then been studied intensively [92, 94, 104], in particular to design codes (*i.e.* graphs) that would behave in the best possible

way when decoded with this algorithm [20, 93]. The idea of approximate inference may look strange in the algebraic context of this chapter, but it had an extraordinary impact in the field of digital communications, as well as in signal and image processing. Let us simply mention the joint source-channel decoding strategies [57], or more generally all the iterative algorithms to jointly perform synchronization, equalization, detection, user separation, decoding, etc. The local extendibility that we mentioned in this chapter is an algebraic generalization of a local optimality criterion discovered by Weiss [108]. There exist several extensions/generalizations of turbo algorithms. For example [113, 114], inspired from statistical physics, the mixture of turbo algorithm with particle methods [100, 101], or the combination of spanning trees [99].

Chapter 3

Networks of dynamic systems

There exists an extremely abundant literature related to Discrete Event Systems (DES), and research is performed in different communities : DES of course, but also Computer Science, Artificial Intelligence, Automatics, etc. A large set of papers develop new models and explore relations between model families. Another large body of results is related to specific applications : one could mention supervisory control (initiated by the pioneering work of Ramadge and Wonham), optimal control, fault diagnosis, model checking (in particular for circuit verification), protocol verification, performance evaluation, etc. A powerful trend in most of these domains aims at addressing large systems, which means the development of *modular*, *decentralized* or *distributed* procedures to cope with the combinatorial explosions inherent to large systems [8, 13, 21, 23, 24, 25, 54, 88, 103]. In all cases, the first step consists in modeling the system in a modular manner. Different (generally equivalent) formalisms have been explored, for example communicating automata (with bounded buffers), bounded or safe Petri nets, networks of partially synchronized automata, etc. In this chapter, we choose the last one, for its simplicity and familiarity, in order to avoid the burden of notations and focus attention on the concepts of distributed computations. We also limit ourselves to the distributed diagnosis problem. As we will see, its central difficulty amounts to 1/ finding a compact representation of sets of trajectories for a large system, and 2/ finding an efficient way to compute with these sets. We believe these ideas are actually central to all the problems above.

3.1 Dynamic systems and their compositions

This section examines how ordinary automata can be combined to form networks of dynamic systems. We first use a standard composition operation, the parallel product, that we later generalize into a more complex form of interaction, called the pullback. One can associate several graphical representations to a network of automata, which will make the connection with the previous chapter. For reasons that will appear later, we must keep track of the elementary automata in the composition operation, so we introduce the unusual "multi-clock" feature in the definition of an automaton. These *multi-clock automata* are the formalized version of the tile systems we sketched in the introduction.

3.1.1 A category of multi-clock automata

Multi-clock automaton. An *automaton* is a 4-tuple $\mathcal{A} = (S, T, s^0, \rightarrow)$ composed of a state set S, a transition set T, an initial state $s^0 \in \mathcal{S}$ and a flow relation $\rightarrow \subseteq (S \times T) \cup (T \times S)$ satisfying $\forall t \in T, |{}^\bullet t| = |t^\bullet| = 1$, where ${}^\bullet t$ and $t^\bullet$ represent pre- and post-states, as usual. Observe that $\rightarrow \subseteq S \times T \times S$, so there may exist several transitions between two states[1]. A *labeled automaton* (LA) $\mathcal{A} = (S, T, s^0, \rightarrow, \lambda, \Lambda)$ is an automaton enriched with a labeling function $\lambda : T \rightarrow \Lambda$. This is used below for synchronization purposes.

A *multi-clock automaton* (MCA) $\mathcal{A} = (S, T, s^0, \rightarrow, \chi, I)$ is an automaton enriched with an index set I and an indexing function $\chi : T \rightarrow 2^I$. These indexes will be used to keep track of elementary automata when they are assembled to form larger systems. So $\chi(t)$ represents automata of $\mathcal{A}$ where transition t has an influence, and conversely $\chi^{-1}(i) = \{t \in T : i \in \chi(t)\}$ identifies transitions of component i (see the "product" subsection).

Naturally, *labeled multi-clock automata* (LMCA) enjoy the two extra functions above. When dealing with several automata $\mathcal{A}_1, \mathcal{A}_2$, etc., we shall use subscripts to identify their elements, like S_i, T_i, etc.

Fig. 3.1 gives two examples of MCA. The numbers close to transition names represent the values of χ, so the MCA on the left "contains" two automata, and for example t_1 belongs to both while t_2 only operates in the first one.

Morphism. Many constructions and results we use in the sequel can be more easily derived in the setting of category theory. The latter can be regarded as a language to study objects and their relations. In our case, the objects will be MCA (or LMCA), and relations *i.e. morphisms* between them are defined as run preserving mappings.

Specifically, let $\mathcal{A}_1, \mathcal{A}_2$ be two MCA, $\mathcal{A}_i = (S_i, T_i, s_i^0, \rightarrow_i, \chi_i, I_i)$, a *morphism* $\phi : \mathcal{A}_1 \rightarrow \mathcal{A}_2$ is a triple (ϕ_S, ϕ_T, ϕ_I) where

1. $\phi_S : S_1 \rightarrow S_2$ is a total function on states, satisfying $\phi(s_1^0) = s_2^0$,
2. $\phi_T : T_1 \rightarrow T_2$ is a partial function on transitions, where $\phi_T(t_1) = \star$ denotes that ϕ_T is undefined at t_1,
3. if $\phi_T(t_1) = \star$, then $\phi_S({}^\bullet t_1) = \phi_S(t_1^\bullet)$, *i.e.* one can erase a transition only if pre- and post-states are merged,
4. if $\phi_T(t_1) \neq \star$, then $\phi_S({}^\bullet t_1) = {}^\bullet \phi_T(t_1)$ and $\phi_S(t_1^\bullet) = \phi_T(t_1)^\bullet$, *i.e.* the flow relation is preserved,
5. $\phi_I : I_1 \rightarrow I_2$ is a relation such that its opposite relation $\phi_I^{op} : I_2 \rightarrow I_1$ is a partial function and $\forall t_1 \in T_1,\ \phi_I(\chi_1(t_1)) = \chi_2(\phi_T(t_1))$, with the convention $\chi_i(\star) = \emptyset$.

[1]The pre- and post-state notations are borrowed to the Petri net formalism. In the same way, $\rightarrow \subseteq (S \times T) \cup (T \times S)$ prepares an extension of this formalism to networks of automata, which are actually almost equivalent to safe Petri nets, the formalism in which this theory was initially developed.

Condition 5 is a bit complex in order to allow the duplication of automata indexes, a property required by the product we define later. Condition 5 implies in particular $\phi_T(t_1) = \star \Leftrightarrow \phi_I(\chi_1(t_1)) = \emptyset$. So when $\chi_1(t_1) \cap Dom(\phi_I) = \emptyset$, one has $\phi_T(t_1) = \star$, and conversely. In summary, ϕ erases all transitions that have no influence on the elementary automata with indexes in $Dom(\phi_I)$, or in other words removes *all* transitions *local* to automata $I \setminus Dom(\phi_I)$. This requirement will ensure that ϕ is "clock preserving," or doesn't modify the counting of time in each automaton, as it will become clear in the sequel.

A morphism ϕ is said to be a *folding* when ϕ_T is a total function and $\phi_I = Id$.

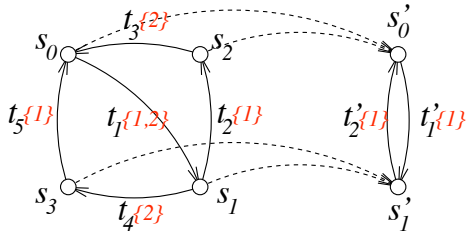


Figure 3.1: *Two multi-clock automata, related by a morphism (dashed arrows).*

Fig. 3.1 gives an example of a morphism $\phi : \mathcal{A}_1 \to \mathcal{A}_2$ relating two MCA. In $\mathcal{A}_1$ (left), one has $\chi_1^{-1}(1) = \{t_1, t_2, t_5\}$ and $\chi_1^{-1}(2) = \{t_1, t_3, t_4\}$, as indicated by the (red) numbers close to transition names. ϕ collapses s_0 and s_2 into s_0', and s_1, s_3 into s_1', and at the same time erases all transitions outside $\chi_1^{-1}(1)$.

For labeled MCA, we introduce some extra requirements in the definition of morphisms: $\phi : \mathcal{A}_1 \to \mathcal{A}_2$ is a morphism of LMCA, with $\mathcal{A}_i = (S_i, T_i, s_i^0, \to_i, \chi_i, I_i, \lambda_i, \Lambda_i)$, iff it is a morphism of MCA and also satisfies

6. $\Lambda_2 \subseteq \Lambda_1$, *i.e.* ϕ reduces the label set,
7. $Dom(\phi_T) = \lambda_1^{-1}(\Lambda_2)$, *i.e.* ϕ is defined on transitions carrying a shared label, and only on them[2],
8. if $\phi_T(t_1) \neq \star$ then $\lambda_1(t_1) = \lambda_2(\phi_T(t_1))$, *i.e.* ϕ preserves labels on its domain of definition $Dom(\phi_T)$.

We denote by $\mathbb{A}$ the category having the LMCA as objects, and the above morphisms as arrows. By abuse of notations, we will write ϕ instead of ϕ_S, ϕ_T or ϕ_I.

3.1.2 Composition by product

The *product* $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ of two LMCA is defined as a standard parallel product: transitions carrying a shared label are synchronized, while transitions carrying a private label remain private (fig. 1.2):

1. $S = S_1 \times S_2$ and $s^0 = (s_1^0, s_2^0)$,

[2]In conjunction with point 5 in the definition of morphisms, this condition means that the labels shared by $\mathcal{A}_1$ and $\mathcal{A}_2$ are exactly associated to the elementary automata preserved by ϕ_I.

2. $T = T_s \cup T_p$ where

$$\begin{aligned} T_s &= \{(t_1, t_2) \in T_1 \times T_2 \,:\, \lambda_1(t_1) = \lambda_2(t_2)\} && (3.1)\\ T_p &= \{(t_1, \star_{s_2}) \,:\, t_1 \in T_1,\, s_2 \in S_2,\, \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} \\ &\quad \cup \{(\star_{s_1}, t_2) \,:\, s_1 \in S_1,\, t_2 \in T_2,\, \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} && (3.2) \end{aligned}$$

   The notation[3] $\star_{s_i}$ stands for a (fake) loop at state s_i.

3. $\to$ is defined by $(s_1, s_2) \to (t_1, t_2) \to (s'_1, s'_2)$ iff $s_i \to_i t_i \to_i s'_i$, $i = 1, 2$, where one of the t_i can be $\star_{s_i}$ and assuming $s_i \to_i \star_{s_i} \to_i s_i$ holds for every state $s_i \in S_i$,

4. $\Lambda = \Lambda_1 \cup \Lambda_2$ and λ follows accordingly,

5. $I = I_1 \uplus I_2$ is the disjoint union of elementary automata indexes, and χ is defined by $\chi(t_1, t_2) = \chi_1(t_1) \uplus \chi_2(t_2)$ with the convention that $\chi_i(\star_{s_i}) = \emptyset$ when $t_i = \star_{s_i}$.

As mentioned in section 3.1.1, the disjoint union of indexes appearing in point 5 allows to keep track of the elementary automata composing an LMCA when a product is performed. This awkward feature incorporated in the basic definition of an automaton will be used to define a vector clock, necessary to the processings we perform later (whence the name "multi-clock automaton").

The label-based product can be considered as a specialization of a more ordinary product defined on MCA. The latter is obtained by removing the label conditions in (3.1) and (3.2) (and also removing point 4, of course). The two products have the same algebraic properties : so labels, which are extremely convenient to build large systems out of components, appear formally as a free feature. Consequently, most proofs can be derived for MCA, for the sake of simplicity, the extension to LMCA being straightforward[4].

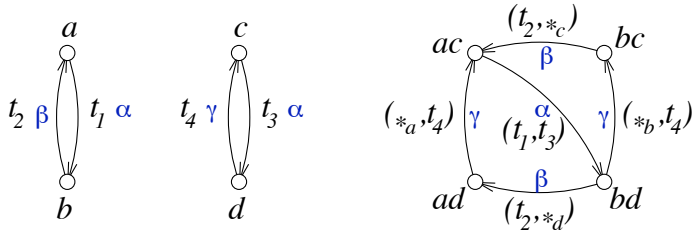


Figure 3.2: *Two LMCA $\mathcal{A}_1, \mathcal{A}_2$ (left) and their label-based product (right). Transition labels are indicated by Greek letters, and we assume $\Lambda_1 \cap \Lambda_2 = \{\alpha\}$.*

Let us denote by $\pi_i : \mathcal{A}_1 \times \mathcal{A}_2 \to \mathcal{A}_i$ the canonical projections from the product $\mathcal{A}_1 \times \mathcal{A}_2$ to their factors $\mathcal{A}_i$. It is straightforward to check that they are morphisms,

[3] This notation avoids introducing stuttering transitions at every state, before computing the product.

[4] This idea is borrowed to [110], which introduces the notion of *synchronization algebra* instead of matching labels.

with the convention that $\pi_1(\star_{s_1}, t_2) = \star$ and symmetrically. Moreover, one has the following property

Proposition 5 (universal property of the product) *Let $\mathcal{A}'$ be an LMCA and let the $\phi_i : \mathcal{A}' \to \mathcal{A}_i,\ i = 1, 2$ be two morphisms. There exists a unique morphism $\psi : \mathcal{A}' \to \mathcal{A}_1 \times \mathcal{A}_2$ such that $\phi_i = \pi_i \circ \psi,\ i = 1, 2$.*

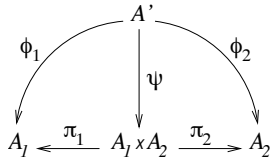


Figure 3.3: *Universal property of the product in $\mathbb{A}$.*

Proposition 5 makes $\times$ the categorical product in $\mathbb{A}$, which immediately entails its associativity. Observe that $\mathcal{A}_1 \times \mathcal{A}_2$ is defined up to a unique isomorphism, as always in category theory, so we will write $\mathcal{A}_1 \times \mathcal{A}_2 = \mathcal{A}$ for any other $\mathcal{A}$ satisfying the universal property (this readily proves the commutativity of $\times$).

Minimal Product covering. Let us define the $\mathcal{A}'_i = \pi_i(\mathcal{A}_1 \times \mathcal{A}_2)$, and $\mathcal{A}' = \mathcal{A}'_1 \times \mathcal{A}'_2$. Then $\mathcal{A}'$ is isomorphic to $\mathcal{A}_1 \times \mathcal{A}_2$, which we also write $\mathcal{A}_1 \times \mathcal{A}_2 = \mathcal{A}'_1 \times \mathcal{A}'_2$.

More generally, let us write $\mathcal{A} \subseteq \mathcal{B}$ when there exists an injective morphism from $\mathcal{A}$ to $\mathcal{B}$. Consider $\mathcal{A} \subseteq \mathcal{A}_1 \times \mathcal{A}_2$, and $\mathcal{A}'_i = \pi_i(\mathcal{A})$, then $\mathcal{A} \subseteq \mathcal{A}'_1 \times \mathcal{A}'_2$, and taking any $\mathcal{A}''_i \subset \mathcal{A}'_i$ instead of $\mathcal{A}'_i$ will break this inclusion. Therefore we call $\mathcal{A}'_1 \times \mathcal{A}'_2$ the *minimal product covering* of $\mathcal{A}$ in $\mathcal{A}_1 \times \mathcal{A}_2$.

Notice that $\mathcal{A} = \mathcal{A}'_1 \times \mathcal{A}'_2$ doesn't hold in general, unless $\mathcal{A}$ already has a product form in $\mathcal{A}_1 \times \mathcal{A}_2$ [5]. This notion of minimal product covering is general and will be used with other categorical products.

3.1.3 Composition by pullback

The product of LMCA provides a simple manner to assemble components by partial synchronization of transitions. However, in some cases, it may be more natural to define interactions by means of shared resources, for example in the case of components communicating by messages.

Specifically, and in order to establish a connection with chapter 2, we would like to define the interaction of two LMCA by the fact that they share some elementary automata, just like two components were sharing variables in chapter 2. We therefore need to define a composition based on the notion of interface, that will enjoy adequate categorical properties. The composition by pullback is a natural candidate.

Consider three LMCA $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$ related by morphisms $f_i : \mathcal{A}_i \to \mathcal{A}_0$, $i \in \{1, 2\}$. This expresses that $\mathcal{A}_0$ is an interface between $\mathcal{A}_1$ and $\mathcal{A}_2$, or that $\mathcal{A}_1, \mathcal{A}_2$ have a common part represented in $\mathcal{A}_0$ (for example a communication buffer or shared

[5] The literature related to languages and their products would say that $\mathcal{A}$ is a *separable* language.

resources in general). The *pullback* of this diagram is defined as a terminal triple $(\mathcal{A}, g_1, g_2)$, with $g_i : \mathcal{A} \to \mathcal{A}_i$, such that $f_1 \circ g_1 = f_2 \circ g_2$. By "terminal," we mean that the following universal property is satisfied (see fig. 3.4) :

$$\begin{array}{c} \forall (\mathcal{A}', h_1, h_2) \ \text{ with } \ h_i : \mathcal{A}' \to \mathcal{A}_i, \\ f_1 \circ h_1 = f_2 \circ h_2 \quad \Rightarrow \quad \exists ! \psi : \mathcal{A}' \to \mathcal{A}, \ h_i = g_i \circ \psi \end{array} \tag{3.3}$$

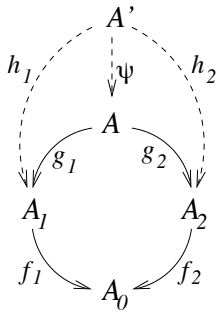


Figure 3.4: *Universal property of the pullback in* $\mathbb{A}$.

Observe that Fig. 3.4 with the null LMCA[6] for $\mathcal{A}_0$ coincides with Fig. 3.2, so the pullback generalizes the product to the case where $\mathcal{A}_1, \mathcal{A}_2$ have a common part. Its detailed definition is thus a bit more complex than the one of the product.

The pullback $\mathcal{A} = \mathcal{A}_1 \overset{\mathcal{A}_0}{\wedge} \mathcal{A}_2$ (or simply $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$ when there is no ambiguity) is given by :

1. $S = \{(s_1, s_2) \in S_1 \times S_2, f_1(s_1) = f_2(s_2)\}$ and $s^0 = (s_1^0, s_2^0)$,
2. $T = T_s \cup T_p$ where

$$\begin{array}{rcl} T_s & = & \{(t_1, t_2) \ : \ f_1(t_1) = f_2(t_2)\} \ \cup \\ & & \{(t_1, t_2) \ : \lambda_1(t_1) = \lambda_2(t_2), \ f_1(t_1) = \star = f_2(t_2), \ f_1({}^\bullet t_1) = f_2({}^\bullet t_2)\} \end{array} \tag{3.4}$$

$$\begin{array}{rcl} T_p & = & \{(t_1, \star_{s_2}) \ : \ \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2, \ f_1(t_1) = \star, \ f_1({}^\bullet t_1) = f_2(s_2)\} \ \cup \\ & & \{(\star_{s_1}, t_2) \ : \ \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1, \ f_2(t_2) = \star, \ f_1(s_1) = f_2({}^\bullet t_2)\} \end{array} \tag{3.5}$$

   (with the convention that t_i ranges over T_i and s_i over S_i),
3. $\to$ is defined by $(s_1, s_2) \to (t_1, t_2) \to (s_1', s_2')$ iff $s_i \to_i t_i \to_i s_i'$, $i = 1, 2$, where one of the t_i can be $\star_{s_i}$ and assuming $s_i \to_i \star_{s_i} \to_i s_i$ holds for every state $s_i \in S_i$,
4. $\Lambda = \Lambda_1 \cup \Lambda_2$ and λ follows accordingly,
5. On indexes, the construction is more technical to account for the structure of the ϕ_I, that must allow index duplications. We only mention it for completeness, but the reader can safely skip it. Consider $I' = I_1 \uplus I_2 \uplus \{\star\}$ and define relation R in I' by

$$i_1 \, R \, i_2 \quad \Leftrightarrow \quad \exists i_0 \in I_0, \ f_1^{op}(i_0) = i_1, \ f_2^{op}(i_0) = i_2 \tag{3.6}$$

[6]The null LMCA has a single state, no transition and an empty index set I.

where i_1 or i_2 can be $\star$. Let $\equiv$ be the equivalence relation generated by R in I', then I is the quotient set $I'/\equiv$ minus the class $\overline{\star}$ (the overline denotes classes). χ is defined by $\chi(t_1,t_2) = \{\overline{i_1} : i_1 \in \chi_1(t_1)\} \cup \{\overline{i_2} : i_2 \in \chi_2(t_2)\}$, and similarly for the other transitions.

The morphisms $g_i : \mathcal{A} \to \mathcal{A}_i$, $i = 1, 2$, are the canonical projections on states and transitions, and on index sets $g_1^{op}(i_1) = \overline{i_1}$, and similarly for g_2^{op}.

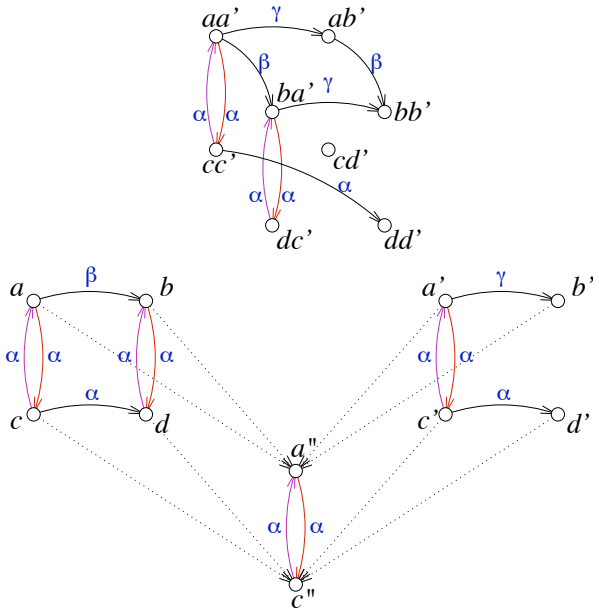


Figure 3.5: *Example of a composition by pullback: the result $\mathcal{A}_1 \wedge \mathcal{A}_2$ is on top, the lowest LMCA is the interface $\mathcal{A}_0$, and the two components $\mathcal{A}_1, \mathcal{A}_2$ appear between them.*

Fig.3.5 illustrates this composition. For the clarity of drawings, only transition labels are mentioned. Morphisms f_1, f_2 are depicted with dotted arrows (only state mappings are represented, transition mappings can be deduced), and morphism g_1, g_2 are suggested by state names (not drawn either). A single label, α, is shared by two LMCA $\mathcal{A}_1, \mathcal{A}_2$. Transitions outside the domains of morphisms f_i synchronize as in an ordinary parallel product.

As for the product, the definition *via* a universal property entails the commutativity of the pullback. The associativity is less natural: the notation $\mathcal{A}_1 \wedge \mathcal{A}_2$ suggests a binary operator, but actually hides a 5-tuple, *i.e.* $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$ plus the two morphisms f_1, f_2. So there is no straightforward meaning to $(\mathcal{A}_1 \wedge \mathcal{A}_2) \wedge \mathcal{A}_3 = \mathcal{A}_1 \wedge (\mathcal{A}_2 \wedge \mathcal{A}_3)$. Nevertheless, the pullback is a categorical *limit*, and the latter can be computed by parts. So if we can give a meaning to $\mathcal{A}_1 \wedge \mathcal{A}_2 \wedge \mathcal{A}_3$, we know that brackets can be introduced. We take another way below to get associativity.

Relations between product and pullback. Let us first simplify the setting to the case we use in the sequel, and justify the utility of the pullback.

Proposition 6 *Consider three LMCA $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2$ and the partial products $\mathcal{A}'_1 = \mathcal{A}_1 \times \mathcal{A}_0$, and $\mathcal{A}'_2 = \mathcal{A}_0 \times \mathcal{A}_2$, with respective canonical projections $f_i : \mathcal{A}'_i \to \mathcal{A}_0$. One has*

$$\mathcal{A}_1 \times \mathcal{A}_0 \times \mathcal{A}_2 \quad = \quad (\mathcal{A}_1 \times \mathcal{A}_0) \overset{\mathcal{A}_0}{\wedge} (\mathcal{A}_0 \times \mathcal{A}_2) \tag{3.7}$$

(3.7) can be derived by a straightforward application of the universal properties of the product and of the pullback (fig. 3.6).

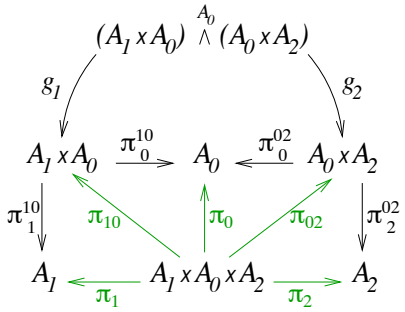


Figure 3.6: *Equivalence between product of three components and pullback of partial products.*

Proposition 6 is important because it allows us to rephrase interactions defined as products in terms of interactions via an interface. Observe however that $\mathcal{A}_0$ does not always capture *all* interactions between $\mathcal{A}_1$ and $\mathcal{A}_2$: the latter can have transitions carrying identical labels, that will thus synchronize in the product, as expressed by (3.4).

Proposition 6 generalizes to the case of N LMCA. Consider $\mathcal{A}_1, \ldots, \mathcal{A}_N$, and for $I \subseteq \{1, \ldots, N\}$ let us define $\mathcal{A}_I \triangleq \times_{i \in I} \mathcal{A}_i$. Then, for $I, J \subseteq \{1, \ldots, N\}$, we define

$$\mathcal{A}_I \wedge \mathcal{A}_J \quad \triangleq \quad \mathcal{A}_I \overset{\mathcal{A}_{I \cap J}}{\wedge} \mathcal{A}_J$$

By (3.7) we have

$$\mathcal{A}_I \wedge \mathcal{A}_J \quad = \quad \mathcal{A}_{I \cup J} \tag{3.8}$$

and consequently

$$\forall I, J, K \subseteq \{1, \ldots, N\}, \qquad (\mathcal{A}_I \wedge \mathcal{A}_J) \wedge \mathcal{A}_K \quad = \quad \mathcal{A}_I \wedge (\mathcal{A}_J \wedge \mathcal{A}_K) \tag{3.9}$$

that we can take as a definition for $\mathcal{A}_I \wedge \mathcal{A}_J \wedge \mathcal{A}_K$. So we are back to a situation where $\wedge$ is a commutative and associative binary operator[7]. $\mathcal{A}_I \wedge \mathcal{A}_J \wedge \mathcal{A}_K$ can actually be defined directly as a categorical limit, but we don't detail this here.

[7] As a rule of thumb, to compute an expression with many pullback signs, one must put brackets around pairs which determines the interfaces to use in each pairwise pullback. The position of brackets doesn't change the result.

3.2 Graphs associated to a multi-clock system

Let us now come back to an LMCA $\mathcal{A}$ defined as a product $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$. Our objective is three-fold :

1. we want to represent graphically all the interactions between the $\mathcal{A}_i$,
2. we need to express these interactions under the form of components interacting by shared variables, in order to use the formalism of chapter 2, and
3. we must obtain a graph on which the separation theorem holds.

There exist different manners to satisfy these conditions ; which one is adequate depends on the choice of composition and projection operators.

Let us call the $\mathcal{A}_i$ *sites*, instead of components, in order to avoid confusions. The interactions between sites are of course due to shared labels. One can adopt several viewpoints to read these interactions. For example, the λ's are resources (variables) that sites (systems) can use to build their trajectories. And sites are of course in competition for the λ's, in the sense that they have different abilities to combine them into sequences. This governs the *direct graph representation*. By contrast, a label λ shared by two or more sites introduces a synchronization constraint between them, in the sense that all these sites must fire synchronously when such a label is concerned. So labels can be considered as local constraint systems defined on subsets of sites, that become the resources (or variables). This governs the *dual graph representation.*

To illustrate these constructions on a running example, let us start from a bipartite graph representation of interactions : The first family of vertices are the sites $\mathcal{A}_i$, the second family is formed by labels λ in $\Lambda = \Lambda_1 \cup \ldots \cup \Lambda_N$, and an edge is drawn between $\mathcal{A}_i$ and λ iff $\lambda \in \Lambda_i$ (Fig. 3.7).

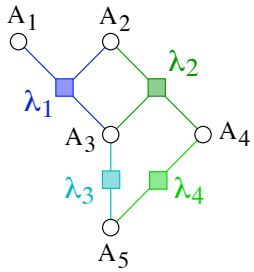


Figure 3.7: *A bipartite graph illustrating the relation between labels and sites* $\mathcal{A}_i$.

3.2.1 Direct graph

Here, labels in $\Lambda = \Lambda_1 \cup \ldots \cup \Lambda_N$ play the part of variables and sites $\mathcal{A}_i$ are the *components* that use them : this corresponds to a setting where system composition is defined by shared labels, and projections are defined with respect to label sets, as for language systems (section 2.1.2). As already detailed in chapter 2, interactions

in $\mathcal{A}$ can be represented as a hypergraph $\mathcal{H}$, where each $\mathcal{A}_i$ defines the hyperedge Λ_i (Fig. 3.8.a).

To form the support of distributed computations, $\mathcal{H}$ must be turned into a connectivity graph $\mathcal{G}^{cnx}$, with sites $\mathcal{A}_i$ as vertices, and where $(\mathcal{A}_i, \mathcal{A}_j)$ is an edge as soon as $\Lambda_i \cap \Lambda_j \neq \emptyset$ (Fig. 3.8.b). The latter in turn must be simplified into a communication graph $\mathcal{G}^c$, by recursively removing redundant edges. An edge $(\mathcal{A}_i, \mathcal{A}_j)$ is *redundant* in a graph $\mathcal{G}$ iff there exists another path $(\mathcal{A}_i, \mathcal{A}_{k_1}, \mathcal{A}_{k_2}, \ldots, \mathcal{A}_{k_L}, \mathcal{A}_j)$ in $\mathcal{G}$ such that $\Lambda_i \cap \Lambda_j \subseteq \Lambda_{k_l}$ for every k_l, $1 \leq l \leq L$ (Fig. 3.8.c).

Naturally, some of the $\mathcal{A}_i$ can be grouped into larger components if a tree-shaped communication graph is needed : on the example, one can replace $\mathcal{A}_3$ and $\mathcal{A}_4$ by a unique node corresponding to $\mathcal{A}_3 \times \mathcal{A}_4$. Grouping $\mathcal{A}_4$ with $\mathcal{A}_5$ would work as well.

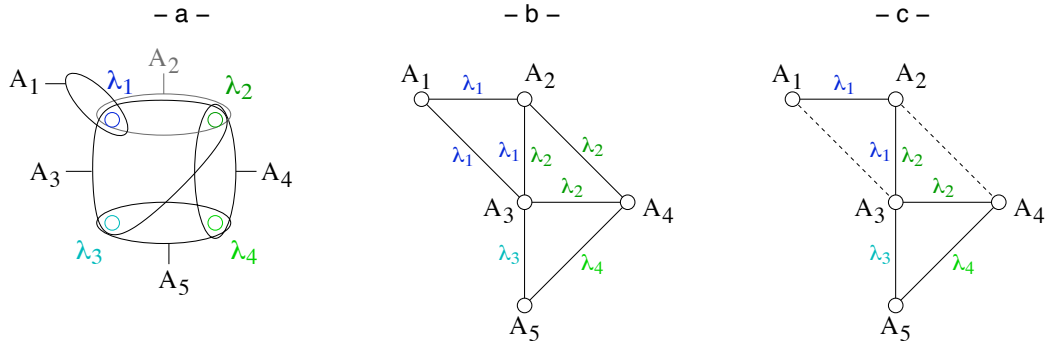


Figure 3.8: *Hypergraph defined by the $\mathcal{A}_i$ on the label set Λ (left), its associated connectivity graph between sites $\mathcal{A}_i$ (center), and a communication graph (right).*

Separation property. Anticipating a little on future sections, this is how the communication graph $\mathcal{G}^c$ between sites will be used. Let $I, J, K \subseteq \{1, \ldots, N\}$ be site index sets. We denote by $I|J|K$ the separation property on $\mathcal{G}^c$: there is no path from a node of I to a node of K that doesn't cross J. $I|J|K$ entails that the MC automata $A_I, \mathcal{A}_J, \mathcal{A}_K$ are such that $\Lambda_I \cap \Lambda_K \subseteq \Lambda_J$, so the common "variables" (*i.e.* labels) of $\mathcal{A}_I$ and $\mathcal{A}_K$ are captured by $\mathcal{A}_J$. Since the latter sees all interactions between $\mathcal{A}_I$ and $\mathcal{A}_K$, one will be able to prove a relation like

$$\Pi_{\Lambda_J}(\mathcal{E}_I \times_T \mathcal{E}_K) \quad = \quad \Pi_{\Lambda_J}(\mathcal{E}_I) \times_T \Pi_{\Lambda_J}(\mathcal{E}_K) \tag{3.10}$$

where $\mathcal{E}_I$ and $\mathcal{E}_K$ will be trajectory sets attached to $\mathcal{A}_I$ and $\mathcal{A}_K$, $\times_T$ a composition operator on them, and Π_{Λ_J} a projection operator on labels. This equation corresponds to axiom (a3) and to a direct application of results presented in chapter 2.

3.2.2 Dual graph

This representation corresponds to the case where projections on labels are impossible and must be replaced by projections on the $\mathcal{A}_i$. In other words, the sites $\mathcal{A}_i$ must be interpreted as the variables. And we must as well define a notion of component, and a composition operation based on shared variables, *i.e.* shared sites. This is where the composition by pullback becomes useful.

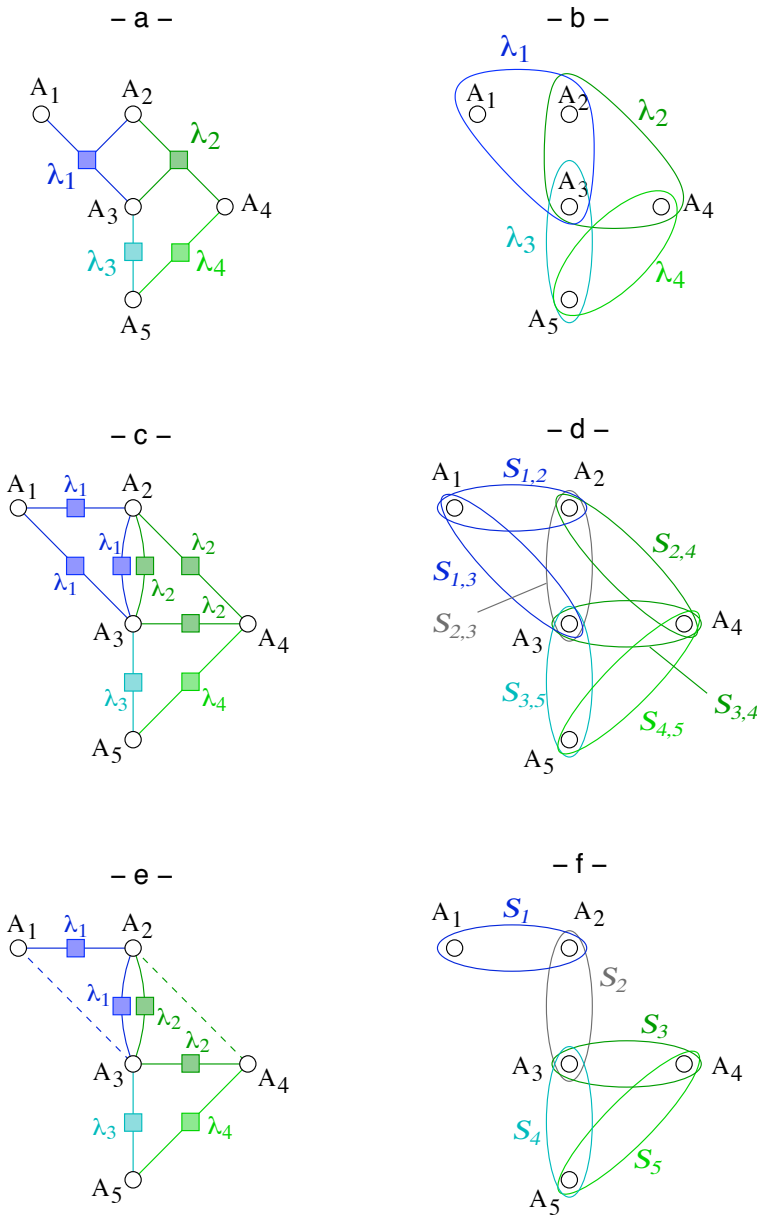


Figure 3.9: *Different ways of defining components as local products of sites $\mathcal{A}_i$. The objective is to capture by such components the constraints that remain in a communication graph on the $\mathcal{A}_i$, as in Fig. 3.8.c.*

Let us define *components* $\mathcal{S}_k = \times_{i \in I_k} \mathcal{A}_i$ where $I_k \subseteq \{1, \ldots, N\}$. By (3.8), one has $\mathcal{A} = \bigwedge_k \mathcal{S}_k$ as soon as the $\mathcal{S}_k$ cover all the $\mathcal{A}_i$. However, this leaves a lot of flexibility.

Considering that every $\lambda \in \Lambda$ introduces a constraint on some of the variables $\mathcal{A}_i$, one can for example take as components the $\mathcal{S}_\lambda$ given by

$$\forall \lambda \in \Lambda, \qquad I_\lambda = \{1 \leq i \leq N \,:\, \lambda \in \Lambda_i\} \quad \text{and} \quad \mathcal{S}_\lambda = \times_{i \in I_\lambda} A_i \tag{3.11}$$

in order to cover the full constraint graph of $\mathcal{A}$. This choice is illustrated in figures 3.9.a and 3.9.b.

This first definition, however, results in quite large components, which can be inefficient for the modular computations described in chapter 2. Another possibility is thus to consider only pairwise synchronizations and take

$$\forall i, j \in \{1, \ldots, N\},\ i < j, \qquad \mathcal{S}_{i,j} = \begin{cases} \mathcal{A}_i \times \mathcal{A}_j & \text{if } \Lambda_i \cap \Lambda_j \neq \emptyset \\ \mathit{null} & \text{otherwise} \end{cases} \tag{3.12}$$

$\mathcal{A} = \bigwedge_{1 \leq i < j \leq N} \mathcal{S}_{i,j}$ is satisfied as soon as every site shares a label with at least one component. This choice is illustrated in Figs. 3.9.c and 3.9.d. In this formulation, the hypergraph induced by components $\mathcal{S}_{i,j}$ on nodes $\mathcal{A}_i$ reduces to an ordinary graph : the latter is nothing more than the connectivity graph $\mathcal{G}^{cnx}$ between sites $\mathcal{A}_i$ (compare Fig. 3.9.d to Fig. 3.8.b). To get the connectivity graph between *components* $\mathcal{S}_{i,j}$, we thus have to take the *dual* of $\mathcal{G}^{cnx}$.

Apparently, this second definition of components involves a larger number of smaller components, and generates a more complex graph. So, at this point, there is no obvious gain. Recall that some edges in $\mathcal{G}^{cnx}$ are redundant, *i.e.* describe synchronization constraints between the $\mathcal{A}_i$ that are captured by others paths. So we reduce the set of $\mathcal{S}_{i,j}$ to those that cover a communication graph $\mathcal{G}^c$ between the $\mathcal{A}_i$ (Fig. 3.9.e and 3.9.f). The remaining components still cover all the $\mathcal{A}_i$, and their connectivity graph is still the dual of $\mathcal{G}^c$. This choice not only reduces the number of components, but also satisfies nice separation properties (see below). In particular, if $\mathcal{G}^c$ is a tree, then every communication graph between the selected $\mathcal{S}_{i,j}$ is a tree.

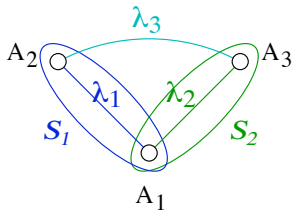


Figure 3.10: *Two components $\mathcal{S}_1 = \mathcal{A}_1 \times \mathcal{A}_2$ and $\mathcal{S}_2 = \mathcal{A}_1 \times \mathcal{A}_3$ that share only variable/site $\mathcal{A}_1$, although their interaction is due both to $\mathcal{A}_1$ and to the "external" label λ_3, not captured by $\mathcal{A}_1$.*

Separation Property. Anticipating again on future sections, this is how an expression like $\mathcal{A} = \bigwedge_{k \in K} \mathcal{S}_k$ will be used, assuming that the $\mathcal{S}_k$ cover a communication graph $\mathcal{G}^c$ between sites $\mathcal{A}_i$.

The first important thing to notice is that the shared variables (= sites) between two $\mathcal{S}_k$ do not capture *all* interactions between these components, so we are not exactly in the setting of chapter 2. Consider the example in Fig. 3.10 : $\mathcal{S}_1$ and $\mathcal{S}_2$ share only variable/site $\mathcal{A}_1$, but they also interact by label λ_3 which is not seen by $\mathcal{A}_1$. This "external" interaction corresponds to the second line of (3.4) in the definition of a pullback, and kills axiom (a3). In other words, it is not possible to directly translate interactions by shared labels into interactions by shared sites.

Nevertheless, there exists a reasonable range of situations where axiom (a3) holds. For K an index set, let us denote $\mathcal{S}_K = \bigwedge_{k\in K} \mathcal{S}_k = \mathcal{A}_{K'}$ where $K' = \cup_{k\in K} I_k$. Consider the aggregated components $\mathcal{S}_I = \mathcal{A}_{I'}$ and $\mathcal{S}_K = \mathcal{A}_{K'}$, their shared variables $\mathcal{A}_{J'}$ (with $J' = I' \cap K'$) *capture all interactions* between $\mathcal{S}_I$ and $\mathcal{S}_K$ iff $\Lambda_{I'} \cap \Lambda_{K'} \subseteq \Lambda_{J'}$[8]. Under this assumption, one will be able to write

$$\Pi_{\mathcal{A}_{J'}}(\mathcal{E}_I \wedge_T \mathcal{E}_K) \quad = \quad \Pi_{\mathcal{A}_{J'}}(\mathcal{E}_I) \wedge_T \Pi_{\mathcal{A}_{J'}}(\mathcal{E}_K) \tag{3.13}$$

for $\mathcal{E}_I$ and $\mathcal{E}_K$ trajectory sets of $\mathcal{S}_I$ and $\mathcal{S}_K$, respectively, and $\wedge_T$ a composition function on these trajectory sets.

Lemma 1 *Assume components $\mathcal{S}_k$ cover a communication graph $\mathcal{G}^c$ of the $\mathcal{A}_i$. Let $\mathcal{S}_J$ separate $\mathcal{S}_I$ from $\mathcal{S}_K$ on their own communication graph, then variables $\mathcal{A}_{J'}$ separate $\mathcal{A}_{I'}$ from $\mathcal{A}_{K'}$ on $\mathcal{G}^c$, and shared variables of $\mathcal{S}_I \wedge \mathcal{S}_J$ and $\mathcal{S}_J \wedge \mathcal{S}_K$ capture all interactions between these aggregated components (Fig. 3.11).*

This limited validity range of axiom (a3) is actually sufficient to rederive all results of chapter 2, in particular the message passing algorithms.

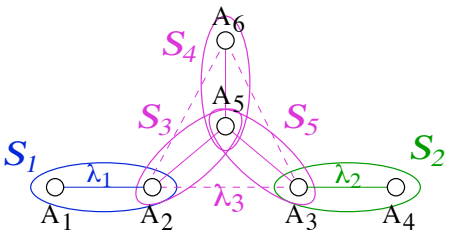


Figure 3.11: *Component $\mathcal{S}_4$ separates $\mathcal{S}_1$ from $\mathcal{S}_2$: its variables see label λ_3, responsible of a direct interaction between $\mathcal{S}_1$ and $\mathcal{S}_2$. Consequently, common variables of $\mathcal{S}_1 \wedge \mathcal{S}_4$ and $\mathcal{S}_4 \wedge \mathcal{S}_2$ capture all interactions between these aggregated components.*

3.3 Diagnosis problem

As mentioned in the introduction of this chapter, we concentrate on diagnosis problems, in a wide sense. We first state the problem, then examine two approaches to solve its centralized version, and their ability to be extended into distributed techniques.

[8]This situation is referenced as the "structural assumption" in [37, 40].

3.3.1 Semantics

So far we have focused on the static structure of systems and didn't use at all the fact that they are *dynamic* systems. To introduce the time dimension, we must give a meaning to the notion of trajectory of an LMCA. For a matter of simplicity, we adopt the usual sequential semantics, also called interleaving semantics, and define runs as sequences of transitions. The next chapters will be based on true concurrency semantics, where runs are partial orders of events. The latter are more adapted to distributed or modular systems, with sparse interactions and thus a high level of concurrency.

As we will see at the end of the chapter, sequential semantics do not fully take advantage of the distributed approach to monitoring problems. But their simplicity allows us to present the mathematical skeleton of this framework and the mechanics of computations, without the technicalities due to partial orders. Once the principles are established, it will be sufficient to check a few key results to adapt this setting to partial order semantics.

Let $\mathcal{A} = (S, T, s^0, \to, \chi, I, \lambda, \Lambda)$ be an LMCA. In the sequential semantics (SS), a *run* (or *trajectory*) of an LMCA $\mathcal{A}$ is modeled as a sequence $\sigma = (t_1, t_2, \ldots, t_N)$ of transitions such that ${}^\bullet t_1 = s^0$ and $t_n^\bullet = s_n = {}^\bullet t_{n+1},\ 1 \leq n \leq N-1$ (also denoted $s^0[t_1\rangle \ldots s_{n-1}[t_n\rangle s_n[t_{n+1}\rangle s_{n+1} \ldots [t_N\rangle s_N$, and $s^0[\sigma\rangle s_N$ for sequences).

3.3.2 Objectives

Centralized diagnosis, single sensor. Consider an LMCA $\mathcal{A}$, and assume $\mathcal{A}$ produces a hidden run σ. One gets information about this run by means of a "sensor" that observes part of the transition labels produced by σ. Specifically, we assume the sensor only collects labels of $\Lambda' \subseteq \Lambda$, and yields a sequence $\mathcal{O}^b$ of observed labels.

The traditional diagnosis problem is stated as follows (see Sampath *et al.* [95]) : assume a subset $T' \subseteq T$ of transitions represents faults that can occur in $\mathcal{A}$. Given $\mathcal{O}^b$, one would like to determine which of the following statements holds :

- a transition of T' was fired for sure in σ,
- σ didn't use any transition of T', for sure,
- σ may have fired a transition of T', $\mathcal{O}^b$ doesn't allow to decide.

There exist more elaborate forms of the problem, that check whether a complex property is satisfied by the hidden run σ given $\mathcal{O}^b$, for example the occurrence of t_1 followed by two occurrences of t_2, etc. (see [61]). They are identical in nature and do not introduce extra formal difficulties.

A more general version of the diagnosis problem consists in recovering all runs of $\mathcal{A}$ that could explain the observed sequence $\mathcal{O}^b$, among which will lie the true hidden run σ. Recovering all runs matching $\mathcal{O}^b$, or estimating the final state of σ given $\mathcal{O}^b$ are almost identical problems. And the latter, called the construction of an *observer* for $\mathcal{A}$, is the main building block in the derivation of a diagnoser for $\mathcal{A}$ (see next sub-section). So we focus on this version here, that we call the *single sensor centralized diagnosis problem*.

Centralized diagnosis, several sensors. In the case of a modular system $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_K$ representing a large distributed machine, it is more reasonable to assume several sensors instead of a single one. These sensors can be attached to the $\mathcal{A}_i$, or equivalently to components $\mathcal{S}_k$. We consider the first case for simplicity, a sensor is attached to each $\mathcal{A}_i$ and observes labels of $\Lambda'_i \subseteq \Lambda_i$ ($\Lambda'_i = \emptyset$ represents the absence of sensor). When $\mathcal{A}$ performs the hidden run σ, the labels produced by transitions operating in $\mathcal{A}_i$ are collected by sensor i under the form of a sequence $\mathcal{O}_i^b$. So the observation is now a tuple $(O_1, \ldots, O_N)$ of sequences, and the objective remains the same : compute all runs of $\mathcal{A}$ that explain these observations. Notice that we lose information with respect to the single sensor case : the exact interleaving of the different $\mathcal{O}_i^b$ is *lost*.

Remark : we assume here that labels attached to transitions in the $\mathcal{A}_i$ play a double role. They are first responsible for transition synchronizations, and they also correspond to observations collected by the sensors. These roles can be separated, for example by introducing an extra label on transitions to define their visible signature. An alternate solution consists in using a more elaborate synchronization algebra between components. In any case, these extensions complexify notations but change very little in the formal aspects of the problem. So we prefer to keep the simplest formulation.

Distributed diagnosis. The previous task can be extremely complex : one must consider all possible interleavings of the O_i, and for each of them the centralized diagnosis problem requires to handle a possibly huge system $\mathcal{A}$. The distributed (or modular) diagnosis approach takes advantage of the interaction structure of $\mathcal{A}$ to avoid these difficulties, provided interactions are sparse enough.

In its simplest form, the distributed approach is based on the following ideas :

- a local supervisor $\mathcal{D}_i$ is attached to each $\mathcal{A}_i$,
- each $\mathcal{D}_i$ has limited knowledge about $\mathcal{A}$ (typically the model of $\mathcal{A}_i$ plus some interface information with the other $\mathcal{A}_j$),
- each $\mathcal{D}_i$ only knows local observations $\mathcal{O}_i^b$,
- the $\mathcal{D}_i$ compute their local view of the global diagnosis, *i.e.* they compute trajectories of their component $\mathcal{A}_i$ that explain local observations $\mathcal{O}_i^b$ and at the same time are coherent with explanations provided by the other local supervisors.

The next sections of this chapter will formalize this approach and analyze its properties. Let us just mention that, ideally, a distributed/modular approach to the diagnosis problem must be equivalent to the centralized one, in the sense that the combination of local results should yield the solution of the (multi-sensor) centralized diagnosis problem.

Historically. Considering the evolution of ideas, several intermediate steps were explored before the distributed diagnosis problem was stated in the above terms.

Early contributions considered first the case of several sensors $\mathcal{O}_i^b$ on the same machine $\mathcal{A}$, and proposed local supervisors making decisions with the knowledge of $\mathcal{A}$ and $\mathcal{O}_i^b$. These agents would then forward their results to some central supervisor in charge of assembling them according to various simple aggregation rules or policies [25, 115]. Despite the expensive use of $\mathcal{A}$, there was no reason why such decentralized approaches should be equivalent to the centralized one. Whence notions of decentralized observability, controllability, diagnosability, etc. Another approach considered distributed observations on a modular model, *i.e.* local agents taking decisions with the knowledge of $\mathcal{A}_i$ and $\mathcal{O}_i^b$, and making assumptions on the interactions with neighboring components. Once again, local agents would forward their results to a central supervisor in charge of assembling them, now with the aim of obtaining equivalence with the centralized diagnosis [8, 88]. The last step was then to get rid of the central supervisor, which meant 1/ distributing its job under the form of a cooperation between local agents, and more essentially 2/ abandoning the objective of global solutions available somewhere, now replaced by a distributed knowledge [103].

3.3.3 The diagnoser approach

Given an LMCA $\mathcal{A}$ and a subset of visible labels $\Lambda' \subseteq \Lambda$, an *observer* for $\mathcal{A}$ is a deterministic automaton $\mathcal{D}_\mathcal{A}$, taking labels of Λ' as input, and which states are subsets of states of $\mathcal{A}$. When $\mathcal{A}$ performs a hidden run σ and outputs the label sequence $\mathcal{O}^b \in \Lambda'^\star$, $\mathcal{D}_\mathcal{A}$ fed with $\mathcal{O}^b$ reaches a unique state that lists all possible states where $\mathcal{A}$ could be, given $\mathcal{O}^b$ (Fig. 3.12).

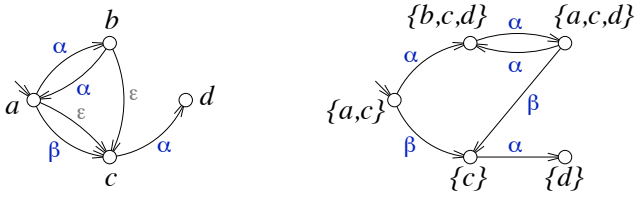


Figure 3.12: *An automaton (left) and its observer (right) for visible labels* $\Lambda' = \{\alpha, \beta\}$.

$\mathcal{D}_\mathcal{A}$ can be obtained easily in two steps [15]:

1. an epsilon-reduction, that replaces each state of $\mathcal{A}$ by its invisible reach (states reachable through invisible transitions), followed by
2. a determinization, that aggregates states reachable by the same transition label.

To obtain a diagnoser for $\mathcal{A}$, let us consider a trivial two state automaton $\mathcal{B}$ that stays at state "green" as long as transitions of $T \setminus T'$ are fired, and otherwise moves from "green" to "red" and stays there forever. The synchronous product $\mathcal{A} \times \mathcal{B}$ (synchronization on transition names) replicates $\mathcal{A}$ with an augmented state, to memorize the firing of a fault transition. The observer of $\mathcal{A} \times \mathcal{B}$ yields a diagnoser for $\mathcal{A}$: when all states of $\mathcal{D}_{\mathcal{A}\times\mathcal{B}}$ are green, the answer will be that no faulty transition

occurred, etc. Notice that by a straightforward extension, one can actually test any regular property on runs of $\mathcal{A}$, *i.e.* properties that can be expressed under the form of an automaton $\mathcal{B}$ [61].

There is a simple case where this construction extends to modular LMCA. Assume $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$, the observer $\mathcal{D}_{\mathcal{A}_1 \times \mathcal{A}_2}$ factorizes as $\mathcal{D}_{\mathcal{A}_1} \times \mathcal{D}_{\mathcal{A}_2}$ when shared labels are visible on both sides, *i.e.* when $\Lambda_1 \cap \Lambda_2 = \Lambda'_1 \cap \Lambda'_2$. Indeed, under this assumption the epsilon-reductions only remove local transitions, and the product of deterministic automata remains deterministic. Interestingly, the visibility of interactions was often assumed in publications related to distributed diagnosis [25], but it doesn't seem its importance for the modularity of the diagnoser has been noticed. The result is not mentioned in [22] for example, although it is fundamental for the modular diagnosability tests developed in this work. And, similarly, the same assumption is hidden in [54, 55]. The extension of the modularity property of the diagnoser to the case of invisible synchronization events remains an open question. The result seems to hold with weaker assumptions, but if modular diagnosers always exist, it is likely that one could derive from them a joint observability/diagnosability test, in the case of distributed observations, which is shown in [105] to be undecidable.

To summarize, diagnosers offer the advantage to be computable off-line : the actual observed sequence is not needed. Diagnosers are "recursive" in nature, in the sense that they can process observations on the fly, and at low cost. On the side of drawbacks lies the problem of size, of course, since they are designed to process any possible sequence of observations. And the modularity of diagnosers doesn't hold for most interesting cases, where component interactions are hidden.

3.4 Distributed diagnosis : the language approach

Distributed computations based on languages were independently proposed by Rong Su in [102, 103]. This approach probably forms the simplest framework for the methodology we propose in this document, so we re-express it here in our formalism. We then adapt it to perform computations on runs of a composite system.

3.4.1 Diagnosis in terms of languages

Centralized diagnosis. Consider a labeled automaton $\mathcal{A} = (S, T, s^0, \rightarrow, \lambda, \Lambda)$ producing a hidden run σ that is observed through the label set $\Lambda' \subseteq \Lambda$. This yields the observed sequence $\mathcal{O}^b = \Pi_{\Lambda'}(\sigma)$ where $\Pi_{\Lambda'}$ is the natural projection on labels of Λ'. Let us denote by $\mathcal{L}(\mathcal{A}) \subseteq T^\star$ the language of $\mathcal{A}$, *i.e.* the set of projections $\Pi_\Lambda(\sigma)$ where σ ranges over all runs of $\mathcal{A}$. In this section, we consider a simplified version of the diagnosis problem : we wish to recover all words of $\mathcal{L}(\mathcal{A})$ that yield $\mathcal{O}^b$ when projected on Λ'. We call this the *diagnosis* of $\mathcal{O}^b$. It is given by

$$\mathcal{D} \;=\; \mathcal{L}(\mathcal{A}) \cap \Pi^{-1}_{\Lambda'}(\mathcal{O}^b) \;=\; \mathcal{L}(\mathcal{A}) \times_L \mathcal{O}^b \tag{3.14}$$

where $\times_L$ is of course the parallel product of languages.

Several sensors. Let us now consider a modular automaton $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$, with $\mathcal{A}_i = (S_i, T_i, s_i^0, \rightarrow_i, \lambda_i, \Lambda_i)$, and distributed observations $\mathcal{O}_i^b$ of the hidden run σ, $\mathcal{O}_i^b = \Pi_{\Lambda_i'}(\sigma)$ where $\Lambda_i' \subseteq \Lambda_i$. As explained in the previous section, distributed observations mean a loss of information : one only knows that there exists an interleaving of the $\mathcal{O}_i^b$ that corresponds to the hidden run σ. So let's take

$$\mathcal{O}^b \quad = \quad \mathcal{O}_1^b \times_L \ldots \times_L \mathcal{O}_N^b \tag{3.15}$$

as observation, which computes all compatible interleavings of the $\mathcal{O}_i^b$. The diagnosis is again given by $\mathcal{D} = \mathcal{L}(\mathcal{A}) \times_L \mathcal{O}^b$, *i.e.* words of $\mathcal{L}(\mathcal{A})$ compatible with at least one interleaving of the $\mathcal{O}_i^b$. Of course, both $\mathcal{O}^b$ and $\mathcal{L}(\mathcal{A})$ are generally large, so in practice there is no hope to solve the problem in that way.

Distributed diagnosis. Let us consider instead the structure of $\mathcal{A}$. From $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$ we derive

$$\mathcal{L}(\mathcal{A}) \quad = \quad \mathcal{L}(\mathcal{A}_1) \times_L \ldots \times_L \mathcal{L}(\mathcal{A}_N) \tag{3.16}$$

Inserting (3.16) and (3.15) into (3.14) yields

$$\mathcal{D} \quad = \quad [\mathcal{L}(\mathcal{A}_1) \times_L \mathcal{O}_1^b] \times_L \ldots \times_L [\mathcal{L}(\mathcal{A}_N) \times_L \mathcal{O}_N^b] \tag{3.17}$$

where $\mathcal{L}(A_i) \times_L \mathcal{O}_i^b \triangleq \mathcal{D}_i$ is a *local diagnosis*, *i.e.* the set of words of component $\mathcal{A}_i$ that explain local observations $\mathcal{O}_i^b$.

We can now apply the results of chapter 2 : variables are the labels in Λ, and components are languages on theses labels (see the language systems section 2.1.2). $\mathcal{D}$ has a product form, where factors $\mathcal{D}_i$ are components operating on labels Λ_i. So $\mathcal{D}$ admits a minimal product covering

$$\mathcal{D} \; = \; \mathcal{D}_1' \times_L \ldots \times_L \mathcal{D}_N' \quad \text{with} \quad \mathcal{D}_i' \; \triangleq \; \Pi_{\Lambda_i}(\mathcal{D}) \tag{3.18}$$

The $\mathcal{D}_i'$ correspond to the *local views of the global diagnosis* : they contain words of $\mathcal{L}(\mathcal{A}_i)$ that both explain local observations $\mathcal{O}_i^b$ and that are compatible with local explanations in all other components.

As explained in chapter 2, the desired $\mathcal{D}_i'$ can be computed in a modular manner without computing $\mathcal{D}$ itself, given the fact that the parallel product of languages $\times_L$ and projections $\Pi_{\Lambda'}$ on subsets of labels satisfy axioms (a1)-(a4). The support of modular computations is a communication graph $\mathcal{G}_c$ between components $\mathcal{A}_i$, in the "direct graph" viewpoint (Fig. 3.8.c), because projections are defined with respect to labels. If $\mathcal{G}_c$ is a tree, convergence to the $\mathcal{D}_i'$ is reached in finite time. Otherwise, one can either group components to recover a tree, or simply ignore cycles and apply a turbo procedure. Since language systems are involutive, turbo algorithms are known to converge to a unique point (at least for the weak topology) and provide reasonable (upper) approximations $\mathcal{D}_i''$ of the $\mathcal{D}_i'$: $\quad \mathcal{D}_i' \subseteq \mathcal{D}_i'' \subseteq \mathcal{D}_i$ and $\mathcal{D} = \mathcal{D}_1'' \times_L \ldots \times_L \mathcal{D}_N''$.

3.4.2 Diagnosis in terms of trajectories

Let us come back to our original problem : determining runs of $\mathcal{A}$ that would explain the sequence $\mathcal{O}^b$ of observed labels.

By post-processing. Let $\dot{\mathcal{L}}(\mathcal{A})$ denote the "language" of $\mathcal{A}$, now expressed in terms of sequences of transitions instead of sequences of labels. Obviously, the new "diagnosis" we are looking for is the inverse projection of $\mathcal{D}$ on $\dot{\mathcal{L}}(\mathcal{A})$:

$$\dot{\mathcal{D}} \triangleq \dot{\mathcal{L}}(\mathcal{A}) \cap \Pi_{\Lambda'}^{-1}(\mathcal{O}^b) = \dot{\mathcal{L}}(\mathcal{A}) \cap \Pi_{\Lambda}^{-1}(\mathcal{D}) \tag{3.19}$$

where $\Pi_{\Lambda'}^{-1}$ yields sequences of transitions. In the same way, when $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$ the local views of the global diagnosis are as well clearly given by reverse projections of the $\mathcal{D}'_i$ on the $\dot{\mathcal{L}}(\mathcal{A}_i)$

$$\dot{\mathcal{D}}'_i = \dot{\mathcal{L}}(\mathcal{A}_i) \cap \Pi_{\Lambda_i}^{-1}(\mathcal{D}'_i) \tag{3.20}$$

However, to prepare for the techniques we develop in the sequel, it is worth determining directly $\dot{\mathcal{D}}$ and the $\dot{\mathcal{D}}'_i$ by computations on trajectory sets.

By direct computation. We first refine the notion of product $\times_L$, to adapt it to trajectory sets. Consider two LMCA $\mathcal{A}_1, \mathcal{A}_2$ and their product $\mathcal{A}_1 \times \mathcal{A}_2$. There exist natural projections $\Pi_i : \dot{\mathcal{L}}(\mathcal{A}_1 \times \mathcal{A}_2) \to \dot{\mathcal{L}}(\mathcal{A}_i)$: they simply replace transitions (t_1, t_2) by t_i, and erase the latter when $t_i = \star_{s_i}$. Let $\mathcal{T}_1, \mathcal{T}_2$ be subsets of $\dot{\mathcal{L}}(\mathcal{A}_1), \dot{\mathcal{L}}(\mathcal{A}_2)$, respectively, we define their label-based product by

$$\mathcal{T}_1 \times_L \mathcal{T}_2 = \Pi_1^{-1}(\mathcal{T}_1) \cap \Pi_2^{-1}(\mathcal{T}_2) \tag{3.21}$$

Naturally, when $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$ this product preserves

$$\dot{\mathcal{L}}(\mathcal{A}) = \dot{\mathcal{L}}(\mathcal{A}_1) \times_L \ldots \times_L \dot{\mathcal{L}}(\mathcal{A}_N) \tag{3.22}$$

For a matter of homogeneity, we could encode $\mathcal{O}^b$ as a labeled sequence of transitions (as we will do in the sequel) in order to compute its product with $\dot{\mathcal{L}}(\mathcal{A})$. For the sake of simplicity, let us rather extend $\times_L$ to sequences of labels, which allows us to write:

$$\dot{\mathcal{D}} = \dot{\mathcal{L}}(\mathcal{A}) \times_L \mathcal{O}^b \tag{3.23}$$

To express our problem in the setting of chapter 2, we must now define systems, variables and projections of systems on these variables. We take $\mathcal{V}_{max} = \{1, \ldots, N\} \uplus \Lambda$, so compared to the previous section, we add site names to the variable set. The systems we consider are formed by

- words w of Λ^*, and by
- labeled sequences of tuples $(t_i)_{i \in I}$ for some $I \subseteq \{1, \ldots, N\}$, that form valid runs of $\mathcal{A}_I$. We denote them by (σ, Λ_I) to keep track of the label set attached to the run σ of $\mathcal{A}_I$.

So we have two types of sequences, but the systems we define below will contain sequences of a single type. The composition operation $\times_L$ applies to both types of sequences.

Let us now come to projections. For $J \subseteq \{1, \ldots, N\}$ and $\Lambda' \subseteq \Lambda$ we must define $\dot{\Pi}_{J \uplus \Lambda'}$ of the two objects above. On a word w of Λ^*, we take

$$\dot{\Pi}_{J \uplus \Lambda'}(w) \quad \triangleq \quad \Pi_{\Lambda'}(w) \tag{3.24}$$

which erases labels not in Λ'. And on a run (σ, Λ_I) of $\mathcal{A}_I$

$$\dot{\Pi}_{J \uplus \Lambda'}(\sigma, \Lambda_I) \quad \triangleq \quad \begin{cases} (\Pi_{I \cap J}(\sigma), \Lambda_I \cap \Lambda') & \text{if} \quad I \cap J \neq \emptyset \\ \Pi_{\Lambda'}(\sigma) & \text{otherwise} \end{cases} \tag{3.25}$$

So the run σ becomes a "pure" label sequence when no site of I is common to J.

With this encoding, $\times_L$ and $\dot{\Pi}$ satisfy axioms (a1)-(a4). Systems $\dot{\mathcal{L}}(\mathcal{A}_I)$ operate on variables $I \uplus \Lambda_I$ and are involutive. The diagnosis $\dot{\mathcal{D}}$ is given by

$$\begin{array}{rcl} \dot{\mathcal{D}} & = & [\dot{\mathcal{L}}(\mathcal{A}_1) \times_L \mathcal{O}_1^b] \times_L \ldots \times_L [\dot{\mathcal{L}}(\mathcal{A}_N) \times_L \mathcal{O}_N^b] \\ & \triangleq & \dot{\mathcal{D}}_1 \times_L \ldots \times_L \dot{\mathcal{D}}_N \end{array} \tag{3.26}$$

and admits the minimal product form

$$\dot{\mathcal{D}} \; = \; \dot{\mathcal{D}}'_1 \times_L \ldots \times_L \dot{\mathcal{D}}'_N \quad \text{with} \quad \dot{\mathcal{D}}'_i \; \triangleq \; \dot{\Pi}_{i,\Lambda_i}(\dot{\mathcal{D}}) \tag{3.27}$$

Once again, this reduction problem can be computed by MPA, where messages are exchanged between sites $\mathcal{A}_i$ (or *disjoint* groups of sites $\mathcal{A}_I$). The properties are the same as for language systems. Nevertheless, observe a very nice feature of this setting: each variable i is *private* to site $\mathcal{A}_i$, *i.e.* sites only share labels. So the messages exchanged between sites are sequences of labels, while sites actually compute on sequences of transitions.

3.4.3 Extensions and drawbacks

The two approaches above, by language systems or by trajectory systems, can be extended in several ways.

Recursivity. First of all, they can be made *recursive*. Assume the $\mathcal{O}_i^b$ are not given once for all, but correspond to growing sequences of observations, that we encode as $\mathcal{O}_i^b(t)$: $\mathcal{O}_i^b(t) \subseteq \mathcal{O}_i^b(t+1)$, $t \geq 0$. As explained at the end of section 2.2.2, message passing algorithms applied to changing local diagnoses $\dot{\mathcal{D}}_i(t) = \dot{\mathcal{L}}(\mathcal{A}_i) \times_L \mathcal{O}_i^b(t)$ remain valid and preserve their convergence properties, provided observations sequences stabilize. And the $\dot{\mathcal{D}}_i(t)$ can of course be computed recursively.

Distributed optimization. Secondly, one could associate cost functions to words of $\mathcal{L}(\mathcal{A}_i)$ or to local runs of the $\dot{\mathcal{L}}(\mathcal{A}_i)$, and adopt a $(\min, +)$ setting to define the composition. Message passing algorithms would then compute (or approximate) the word (resp. the run) of minimal cost in $\mathcal{L}(\mathcal{A})$ (resp. in $\dot{\mathcal{L}}(\mathcal{A})$) explaining all observations.

Drawbacks. Both aspects can of course be combined, so apparently we already have the appropriate framework to deal with networks of dynamic systems and we could stop here. However, these approaches based on sets of sequences suffer from two serious drawbacks.

1. The first drawback relates to the size of the local diagnoses $\mathcal{D}_i(t)$ or $\dot{\mathcal{D}}_i(t)$: they grow rapidly with the length of the observation sequence $\mathcal{O}_i^b(t)$, because of branchings occurring in $\mathcal{A}_i$. So even if one performs modular computations, the "modules" themselves become enormous with time. We therefore need to find a compact manner to encode sets of runs of a component $\mathcal{A}_i$. This is precisely the objective of the next section.

2. The second drawback is a consequence of the sequential semantics, inappropriate for concurrent systems. It multiplies the number of runs by computing useless interleavings. This effect appears in particular in product operations when few labels are common. Consider the extreme case of two components $\mathcal{A}_1, \mathcal{A}_2$ that do not interact at all (no shared label), and a trajectory σ_i of n events in each of these components. There exist 2^n different interleavings of these two runsσ_1 and σ_2, that will be considered as different runs of the joint system $\mathcal{A}_1 \times \mathcal{A}_2$. Even with a compact way of encoding local diagnoses $\mathcal{D}_i$, combinatorial explosions remain due to the fact that concurrency is not handled properly (see section 3.7). The next chapters will address this difficulty by recasting all results in a true concurrency semantics.

3.5 Trajectory sets in the sequential semantics

The mathematical skeleton of the previous section gives an accurate view of the structure of computations that we perform in the sequel. The main change that we introduce now relates to the objects on which these operations apply : we replace languages by more complex structures that encode sets of runs in a compact manner.

3.5.1 Trellis automaton

We had already defined the category $\mathbb{A}$, formed by labeled multi-clock automata associated to a notion of morphism. A guiding property when one defines morphisms is that they must preserve runs :

Lemma 2 *Let $\phi : \mathcal{A}_1 \to \mathcal{A}_2$ be an LMCA morphism, and let σ_1 be a run of $\mathcal{A}_1$, then $\sigma_2 = \phi(\sigma_1)$ is a run of $\mathcal{A}_2$.*

Naturally, $\phi(\sigma_1)$ is defined by the recursion $\phi(\sigma_1|t_1) = \phi(\sigma_1)|\phi(t_1)$ if $\phi(t_1) \neq \star$, and $\phi(\sigma_1|t_1) = \phi(\sigma_1)$ otherwise.

We now focus on the representation of *sets* of trajectories for a given LMCA. To do so, we introduce a sub-category of LMCA : $\mathcal{A}$ is said to be a *trellis automaton* (TA) when

1. the graph of $\mathcal{A}$, determined by $\to$, is directed and acyclic, or equivalently $\to^*$ defines a partial order on nodes of $\mathcal{A}$ (*i.e.* states and transitions),

2. this partial order is well founded: $\forall x \in S \cup T,\ |\{x' \in S \cup T : x' \rightarrow^* x\}| < \infty$,

3. s^0 is the unique minimum of this partial order: $\forall x \in S \cup T,\ |\{x' \in S \cup T : x' \neq x,\ x' \rightarrow^* x\}| = 0 \Leftrightarrow x = s^0$.

As a consequence, in any run $\sigma = (t_1, \ldots, t_n)$ of a TA, all transitions t_i are different. In the sequel, states of a TA are called *conditions* (denoted by $c \in C$ instead of $s \in S$), and transitions are called *events* (denoted by $e \in E$ instead of $t \in T$)[9].

A trellis automaton $\kappa = (C', E', \rightarrow', c^0, \chi', I, \lambda', \Lambda)$ is called a *configuration* when it is composed of a single sequence of events: $\forall c \in C', |{}^\bullet c| \leq 1$ and $|c^\bullet| \leq 1$. We will say that κ is a "configuration of $\mathcal{T}$" when this configuration is a sub-TA of $\mathcal{T}$. Changing slightly notations, we now identify runs of $\mathcal{T}$ with its configurations.

Multi-clock function. To events of $\mathcal{T}$ we associate a vector-clock value defined by the function

$$\begin{array}{lll} \forall e \in E, & h_e : & I \rightarrow \mathbb{N} \\ & & i \mapsto h_e(i) = \mathbb{1}_{i \in \chi(e)} \end{array} \tag{3.28}$$

So an event counts 1 for component i iff it influences this component. This definition leads to the notion of *height* of a condition c in a configuration κ of $\mathcal{T}$. Assume κ corresponds to the sequence of events $(e_1, e_2, \ldots, e_n, \ldots)$ and goes through condition c immediately after e_n, we define

$$H_\kappa(c) \triangleq \sum_{n=1}^{N} h_{e_n} \tag{3.29}$$

For every component index $i \in I$, $H_\kappa(c)$ thus counts the number of events associated to component i and before c in run κ, whence the name "multi-clock function." It obviously relates to vector-clocks of Mattern [84] and Fidge [50].

A trellis automaton $\mathcal{T}$ is *correctly folded* iff $\forall c \in C$ and for any pair κ, κ' of configurations of $\mathcal{T}$ containing c, one has $H_\kappa(c) = H_{\kappa'}(c)$. In other words, only sequences of events with the same multi-clock value can merge at the same condition c (see Fig. 3.13).

From now, we only consider correctly folded trellis automata; they form the subcategory $\mathbb{T}$ of $\mathbb{A}$. In a (correctly folded) TA $\mathcal{T}$, one can of course write $H(c)$ instead of $H_\kappa(c)$.

3.5.2 Time-unfolding of an automaton

Let $\mathcal{A} \in \mathbb{A}$, $\mathcal{T} \in \mathbb{T}$, and let $f : \mathcal{T} \rightarrow \mathcal{A}$ be a morphism. The pair $(\mathcal{T}, f)$ is a *trellis process* (TP) of $\mathcal{A}$ iff

1. f is a folding of $\mathcal{T}$ onto $\mathcal{A}$ (*i.e.* a total function on states and transitions),

[9] These names come from the definition of branching processes, that describe runs of Petri nets in the true concurrency semantics.

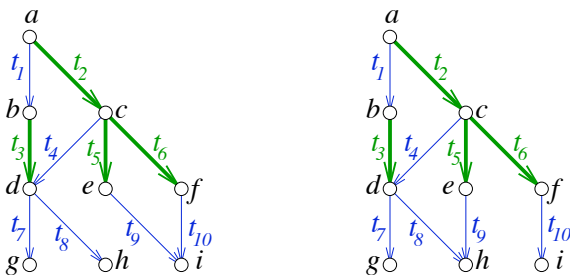


Figure 3.13: *Two trellis automata, with two components : thin arrows represent transitions associated to component 1, and thick arrows those associated to component 2. The TA on the left is correctly folded, the other one isn't : the merge at h is illegal.*

2. $\mathcal{T}$ satisfies the parsimony criterion :

$$\forall e, e' \in E, \qquad [\,{}^\bullet e = {}^\bullet e',\ \ f(e) = f(e')\,] \ \Rightarrow\ e = e' \tag{3.30}$$

3. $\mathcal{T}$ is maximally folded :

$$\forall c, c' \in C, \qquad [\,H(c) = H(c'),\ \ f(c) = f(c')\,] \ \Rightarrow\ c = c' \tag{3.31}$$

f can be viewed as a labeling of events of $\mathcal{T}$ by transitions of $\mathcal{A}$. This definition thus ensures that if κ is a configuration of $\mathcal{T}$, then $(\kappa, f_{|\kappa})$ encodes a run of $\mathcal{A}$ (point 1). Moreover, thanks to the parsimony criterion,

Lemma 3 *Let $(\mathcal{T}, f)$ and (κ, f') be trellis processes of $\mathcal{A}$, and assume κ is a configuration. There exists at most one injective morphism $\phi : \kappa \to \mathcal{T}$ such that $f' = f \circ \phi$.*

This states that a run of $\mathcal{A}$ is represented at most once in a trellis process of $\mathcal{A}$. As a direct consequence,

Lemma 4 *Two trellis processes of $\mathcal{A}$ representing the same trajectories of $\mathcal{A}$ are actually isomorphic.*

So trellis processes provide the compact structure we are looking for to describe sets of runs of an LMCA.

Notation: for the sake of simplicity, we often omit mentioning the folding f in a run (κ, f) or a trellis process $(\mathcal{T}, f)$ of $\mathcal{A}$, and simply talk about κ or $\mathcal{T}$.

Let $\mathcal{A} = (S, T, s^0, \to, \chi, I, \lambda, \Lambda)$ be an LMCA. One can build a trellis process $(\mathcal{T}, f)$ of $\mathcal{A}$ by the following recursion :

Procedure 1

1. Initialization : $C = \{c^0\},\ f(c^0) = s^0,\ E = \emptyset$
2. Recursion : for $c \in C$ and $t \in T$ such that $f(c) = {}^\bullet t$,
   if $\nexists e \in E$ such that ${}^\bullet e = c$ and $f(e) = t$, then
   (a) create $e \in E$, set ${}^\bullet e = c,\ f(e) = t,\ \chi'(e) = \chi(t)$ and $\lambda'(e) = \lambda(t)$,

(b) create $c' \in C$, set ${}^\bullet c' = e$ and $f(c') = t^\bullet$,

(c) if $\exists c'' \in C$ with $f(c'') = f(c')$ and $H(c'') = H(c')$, then merge c' and c''.

The three conditions of the definition are satisfied by construction, and clearly any trellis process of $\mathcal{A}$ can be obtained in that way, with a suitable guiding of events to connect.

On the basis of the above unfolding procedure, one can recursively define the *union* of two or more trellis processes of $\mathcal{A}$, which produces another TP of $\mathcal{A}$. Notice right now that the union $\mathcal{T}_1 \cup \mathcal{T}_2$ may contain configurations that were not present in any of the $\mathcal{T}_i$, but that nevertheless remain valid runs of $\mathcal{A}$. We come back on this point later.

Lemma 5 *Let $(\mathcal{T}, f)$ be a trellis process of $\mathcal{A}$, then $\mathcal{T}$ is isomorphic to the union of its configurations (κ_i, f_i). As a consequence, two trellis processes of $\mathcal{A}$ that have isomorphic configurations are isomorphic.*

Let us define the *prefix* relation $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ between TP by the fact that $\mathcal{T}_2$ contains more configurations of $\mathcal{A}$ than $\mathcal{T}_1$, or equivalently that there exists an injective morphism $\phi : \mathcal{T}_1 \to \mathcal{T}_2$. The above lemmas lead to the following important result:

Theorem 7 *Given an LMCA $\mathcal{A}$, there exists a unique maximal trellis process of $\mathcal{A}$ for the prefix relation. We denote it $(\mathcal{U}^{st}_{\mathcal{A}}, f_{\mathcal{A}})$ and call it the* sequential time-unfolding *of $\mathcal{A}$ (ST-unfolding or STU for short).*

The proof is simple: Define $(\mathcal{U}^{st}_{\mathcal{A}}, f_{\mathcal{A}})$ as the union of all trajectories (κ, f) of $\mathcal{A}$. Unicity is obvious from lemma 5, which also shows that the STU contains all trellis processes of $\mathcal{A}$.

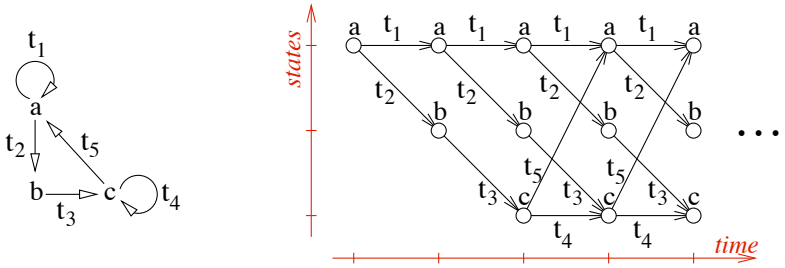


Figure 3.14: *An ordinary automaton A(left) and its sequential time-unfolding $\mathcal{U}^{st}_{\mathcal{A}}$ (right). The folding $f_{\mathcal{A}}$ is represented by state and transition names close to conditions and events.*

Fig. 3.14 illustrates the notions of trellis process and sequential time-unfolding on an ordinary automaton, *i.e.* an LMCA with $|I| = 1$. The STU, commonly called the trellis of an automaton in digital communications, is the support of the Viterbi algorithm, for example in the maximum likelihood decoding of noisy bits when transmissions are protected by a convolutional error correcting codes. Trellises also form the implicit support of the Bellman equation, in optimal control problems. So only the multi-clock aspects are new.

Observe that the size of an ordinary trellis grows linearly with its length. With a multi-clock criterion to determine merge points, the number of possible merges decreases and one obtains a larger structure, with growing width. But it remains smaller than the simple unfolding of $\mathcal{A}$, that unfolds both time and conflicts (or choices) and yields a decision tree. And the latter is in turn smaller than the language of $\mathcal{A}$.

Relation between trellis processes and sub-languages. The time-unfolding of $\mathcal{A}$ encodes all possible runs of $\mathcal{A}$, or equivalently all the language of $\mathcal{A}$. But recall that a union of configurations of $\mathcal{A}$ generally results in a TP that contains *more* runs. So what are the sub-languages of $\mathcal{A}$ that can be represented by a trellis process ?

Let (κ_1, f_1) and (κ_2, f_2) be two runs of $\mathcal{A}$, with $c_i^0[\kappa_i\rangle c_i$. They are said to be H-equivalent, denoted by $\kappa_1 \simeq^H \kappa_2$, iff $f_1(c_1) = f_2(c_2)$ and $H_{\kappa_1} = H_{\kappa_2}$, *i.e.* iff they have same length, and start and finish at identical states of $\mathcal{A}$. A sub-language $\mathcal{L}$ of $\mathcal{A}$ is said to be suffix-closed iff

$$\forall \kappa = \kappa_1\kappa_1' \in \mathcal{L},\ \forall \kappa_2 \in \mathcal{L}, \quad \kappa_1 \simeq^H \kappa_2 \quad \Rightarrow \quad \kappa_2\kappa_1' \in \mathcal{L} \tag{3.32}$$

Lemma 6 *Let $\mathcal{L}$ be a sub-language of $\mathcal{A}$, $\mathcal{L}$ can be encoded as a trellis process of $\mathcal{A}$ iff it is prefix- and suffix-closed.*

The prefix-closure requirement is a little embarrassing, but can be easily discarded. Let us enrich the definition of a trellis process $(\mathcal{T}, f)$ with a *stop function* $St : C \to \{0, 1\}$ on conditions. We then associate to $(\mathcal{T}, f, St)$ the sub-language $\mathcal{L}$ of $\mathcal{A}$ formed by configurations (κ, f) of $\mathcal{T}$ that finish on a stop point : $c^0[\kappa\rangle c$ and $St(c) = 1$. Then

Proposition 7 *There is a one to one correspondence between stopped trellis processes of $\mathcal{A}$ and sub-languages $\mathcal{L}$ of $\mathcal{A}$ that satisfy*

$$\forall \kappa = \kappa_1\kappa_1' \in \mathcal{L},\ \forall \kappa_2 \in \mathcal{L}, \quad \kappa_1 \simeq^H \kappa_2 \quad \Rightarrow \quad \kappa_1 \in \mathcal{L} \ \wedge\ \kappa_2\kappa_1' \in \mathcal{L} \tag{3.33}$$

(3.33) expresses both suffix-closure and a weaker form of prefix-closure.

Notice however that the above limitations in the expressive power of trellis processes will not prevent modular computations and diagnosis based on them.

3.5.3 Variations around the height function

The multi-clock function that governs merge points of trajectories is somehow arbitrary and can be generalized, provided its vector nature is preserved. So far, each event affecting component i was counting for 1 in the clock of that component. One can easily relax this assumption.

For example, let $(\mathcal{E}, \odot, \epsilon)$ be a set provided with an internal composition law $\odot$, and ϵ as neutral element. Let us attach to each LMCA $\mathcal{A}$ a function that assigns a height vector to all transitions : $\forall t \in T,\ h_t : I \to \mathcal{E}$ such that $i \notin \chi(t) \Rightarrow h_t(i) = \epsilon$, so the height of t is neutral for component i when t doesn't affect it. Naturally, we provide these new LMCA with height preserving morphisms.

To define the height of a condition c in a configuration κ, we simply combine these height values : for κ terminating at c and consisting of events $(e_1, \ldots, e_n)$, we take $H_\kappa(c) = h_{e_1} \odot \ldots \odot h_{e_n}$ (component-wise composition of vectors).

One can of course choose $(\mathcal{E}, \odot, \epsilon)$ in order to have monotonic clocks, but this is not necessary : circuits apparently don't bother the theory[10]. They can even be useful to avoid unfolding specific parts of a system. Assume for example that one wishes to count events that produce a label in $\Lambda' \subset \Lambda$ and ignore all other events. This can be done by $h_e = \epsilon$ whenever $\lambda(e) \notin \Lambda'$. So silent loops are not unfolded in the time-unfolding of $\mathcal{A}$. We refer the reader to [46, 47] for the interest of this property in diagnosis applications. Notice that the idea of not unfolding silent loops was introduced by Lamperti and Zanella [72].

By contrast, it is extremely important to preserve the vector nature of a height function. Abandoning this structure means breaking the universal property stated in theorem 8 below, and consequently blocking the distributed diagnosis approach based on it. We come back on the necessity of vector clocks at the end of the chapter (see also [46, 47]).

As a remark in passing, let us mention that one can easily design a strictly increasing height function that would allow no merge point at all. For example a height function that stores all the past of a condition. In this case, the sequential time-unfolding of $\mathcal{A}$ coincides with its ordinary unfolding, also called a *decision-tree*. Consequently, all results we state on sequential time unfoldings remain valid for ordinary unfoldings.

3.5.4 Categorical properties

The awkward function $\chi : T \to I$ introduced in the definition of LMCA could have appeared as a useless decoration up to now. This feature is actually necessary to obtain the following result.

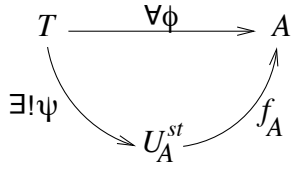


Figure 3.15: *Universal property of the STU of $\mathcal{A}$.*

Theorem 8 (Universal property) *Let $\mathcal{A} \in \mathbb{A}$ be an LMCA, let $\mathcal{T} \in \mathbb{T}$ be a trellis process and $\phi : \mathcal{T} \to \mathcal{A}$ a morphism, there exists a unique morphism $\psi : \mathcal{T} \to \mathcal{U}_\mathcal{A}^{st}$ such that $\phi = f_\mathcal{A} \circ \psi$.*

Theorem 8 essentially expresses that $\mathcal{U}^{st}(\mathcal{A})$ describes all runs of $\mathcal{A}$, and describes them only once (this explains the necessity of being "maximally folded" for a trellis processes).

[10] This is only a conjecture at this point, verified on a set of examples. For a rigorous proof, one would have to redefine trellis processes to allow circuits, under the constraint of a height function. And then check the preservation of theorems 7 and 8.

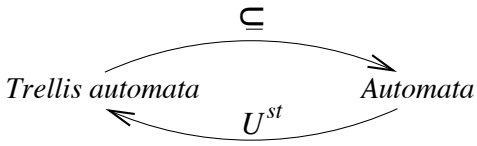


Figure 3.16: *Two functors relating* $\mathbb{A}$ *and* $\mathbb{T}$*, that form an adjunction.*

When designing a structure to encode runs of $\mathcal{A}$, as soon as its universal property is satisfied, some classical category theory arguments are triggered and lead to very useful results. We sketch this reasoning below.

$\mathbb{T}$ being a sub-category of $\mathbb{A}$, there exists an inclusion functor $F = \subseteq : \mathbb{T} \to \mathbb{A}$. Conversely, the ST-unfolding operation on LMCA defines a functor $G = \mathcal{U}^{st} : \mathbb{A} \to \mathbb{T}$. By "functor," we mean that $\mathcal{U}^{st}$ applies also to morphisms and satisfies $\mathcal{U}^{st}(f \circ g) = \mathcal{U}^{st}(f) \circ \mathcal{U}^{st}(g)$. The fact that $G = \mathcal{U}^{st}$ is a functor is a direct consequence of the universal property associated to any $\mathcal{U}^{st}_{\mathcal{A}}$. The latter also entails that (F, G) forms a pair of *adjoint functors* (see [81] chapter 4, in particular theorem 2.iv, p. 81). As the right adjoint of the pair, G preserves categorical limits, and so preserves products (see [81] chapter 5.5, in particular theorem 1, p. 114). In other words, one has

$$\forall \mathcal{A}_1, \mathcal{A}_2 \in \mathbb{A}, \qquad \mathcal{U}^{st}(\mathcal{A}_1 \times \mathcal{A}_2) \;=\; \mathcal{U}^{st}(\mathcal{A}_1) \times_T \mathcal{U}^{st}(\mathcal{A}_2) \tag{3.34}$$

where $\times_T$ denotes the categorical product in $\mathbb{T}$. By contrast with $\times$, the product in $\mathbb{T}$ not only synchronizes events with shared labels, but also preserves the partial ordering of the resulting structure. It must be understood as an operation synchronizing trajectories instead of automata.

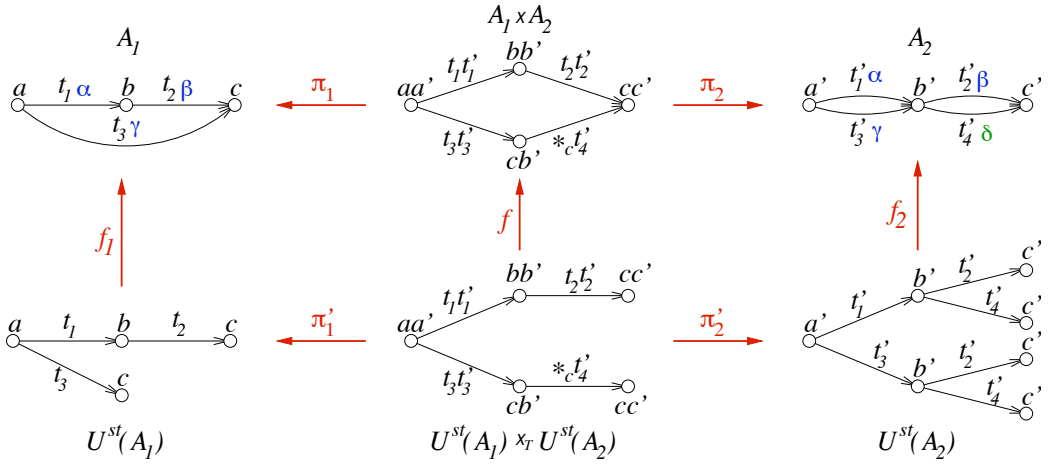


Figure 3.17: *Top : two LMCA and their product. Bottom : the corresponding STU (on this example they coincide with classical unfoldings). The 7 morphisms relating these LMCA form a commutative diagram.*

Fig.3.17 illustrates this property. The top line represents two LMCA $\mathcal{A}_1$ and $\mathcal{A}_2$, with their product (in the center), and the associated canonical projections $\pi_i : \mathcal{A}_1 \times \mathcal{A}_2 \to \mathcal{A}_i$. $\mathcal{A}_1$ and $\mathcal{A}_2$ share labels $\{\alpha, \beta, \gamma\}$ but δ is private to $\mathcal{A}_2$. The

bottom line represents the corresponding sequential time-unfoldings, together with their foldings (materialized by state and transition names close to conditions and events). The STU $\mathcal{U}^{st}_{\mathcal{A}_1\times\mathcal{A}_2}$ is isomorphic to the product $\mathcal{U}^{st}(\mathcal{A}_1)\times_T\mathcal{U}^{st}(\mathcal{A}_2)$, which has canonical projections $\pi'_i:\mathcal{U}^{st}(\mathcal{A}_1)\times_T\mathcal{U}^{st}(\mathcal{A}_2)\to\mathcal{U}^{st}(\mathcal{A}_i)$ to its factors. The 7 morphisms mentioned on the picture form two commutative squares, thanks to theorem 8. Observe that $\mathcal{U}^{st}_{\mathcal{A}_1\times\mathcal{A}_2}$ has two conditions pointing to the same state (c,c') of $\mathcal{A}_1\times\mathcal{A}_2$: they correspond to different multi-clock values, and thus can't be merged. By doing so, one would get a structure isomorphic to $\mathcal{A}_1\times\mathcal{A}_2$, and there wouldn't exist a morphism π'_1 anymore. In other words, that would kill (3.34), which shows the importance of separate counting of time in the various components of an LMCA.

The adjoint pair (F,G), where F corresponds to an inclusion, is called a *coreflection*. Since additionally one has $\forall\mathcal{T}\in\mathbb{T},\ \mathcal{T}\simeq\mathcal{U}^{st}(\mathcal{T})$ (read "isomorphic to"), one can actually define the product $\times_T$ by

$$\forall\mathcal{T}_1,\mathcal{T}_2\in\mathbb{T},\qquad \mathcal{T}_1\times_T\mathcal{T}_2\ \simeq\ \mathcal{U}^{st}(\mathcal{T}_1)\times_T\mathcal{U}^{st}(\mathcal{T}_2)\ =\ \mathcal{U}^{st}(\mathcal{T}_1\times\mathcal{T}_2)\quad(3.35)$$

This last relation has an important practical consequence: by combining Procedure 1, computing ST-unfoldings, to the definition of the product in $\mathbb{A}$, one gets a recursive algorithm to compute the product in $\mathbb{T}$.

Finally, just like products, pullbacks are also categorical limits. So the reasonings above apply in the same way and one has

$$\forall\mathcal{A}_0,\mathcal{A}_1,\mathcal{A}_2\in\mathbb{A},\qquad \mathcal{U}^{st}(\mathcal{A}_1\overset{\mathcal{A}_0}{\wedge}\mathcal{A}_2)\ =\ \mathcal{U}^{st}(\mathcal{A}_1)\overset{\mathcal{U}^{st}(\mathcal{A}_0)}{\wedge_T}\mathcal{U}^{st}(\mathcal{A}_2)\quad(3.36)$$

where the pullback $\wedge_T$ in the sub-category $\mathbb{T}$ of trellis automata is defined by

$$\forall\mathcal{T}_0,\mathcal{T}_1,\mathcal{T}_2\in\mathbb{T},\qquad \mathcal{T}_1\overset{\mathcal{T}_0}{\wedge_T}\mathcal{T}_2\ \simeq\ \mathcal{U}^{st}(\mathcal{T}_1)\overset{\mathcal{U}^{st}(\mathcal{T}_0)}{\wedge_T}\mathcal{U}^{st}(\mathcal{T}_2)\ =\ \mathcal{U}^{st}(\mathcal{T}_1\overset{\mathcal{T}_0}{\wedge}\mathcal{T}_2)\quad(3.37)$$

And once again, the combination of Procedure 1 with the definition of $\wedge$ gives a recursive algorithm to compute pullbacks in $\mathcal{T}$.

3.6 Distributed diagnosis: the trellis approach

The properties we have obtained on trellis processes are now very close to those of language systems, so can we really *compute* with trellises ?

In this section, we come back to our diagnosis application with these new tools. We follow the steps of section 3.4 to highlight the formal similarities, but also the technical extensions that are needed. As we will see, the major difference is that we can't use of projections on subsets of labels. So projections will now be defined with respect to sites $\mathcal{A}_i$, and we will adopt the dual graph viewpoint of section 3.2.2.

3.6.1 Centralized diagnosis, single sensor

The setting doesn't change: $\mathcal{A}$ performs a hidden run κ whose labels are observed under the form of sequence $\mathcal{O}^b$. The latter can be encoded as a trivial trellis automaton, formed of a single sequence, *i.e.* $\mathcal{O}^b$ is a configuration. We are still looking

for runs of $\mathcal{A}$ that would explain $\mathcal{O}^b$. Since we have a product on trellis processes, (3.14) becomes

$$\mathcal{D} \quad = \quad \mathcal{U}^{st}(\mathcal{A}) \times_T \mathcal{O}^b \tag{3.38}$$

A few comments on this equation.

It is meaningless to unfold $\mathcal{A}$ and then compute the product. This formal relation must be translated in practice into a recursive algorithm : one computes the unfolding of $\mathcal{A}$ *guided* by observations $\mathcal{O}^b$, which is closer to the expression $\mathcal{D} = \mathcal{U}^{st}(\mathcal{A} \times \mathcal{O}^b)$ given by (3.35).

Secondly, by contrast with (3.14) or (3.23), $\mathcal{D}$ defined in (3.38) contains runs of $\mathcal{A}$ that explain only a prefix of $\mathcal{O}^b$, and not the entire $\mathcal{O}^b$. This effect can be corrected easily : it suffices to use stopped trellis processes, as defined at the end of section 3.5.2. They form a sub-category of stopped automata, assuming morphisms that preserve stop points. The theory is identical to what has been presented above, so we omit this detail, for the sake of simplicity, and assume that (3.38) does yield runs of $\mathcal{A}$ that entirely explain $\mathcal{O}^b$. We refer the reader to [46, 47] and [48] for details.

3.6.2 Centralized diagnosis, several sensors

When $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$ and $\mathcal{A}_i$ has Λ_i as label set, assume again that labels of the hidden run κ are collected by independent sensors attached to the sites $\mathcal{A}_i$. The sensor on $\mathcal{A}_i$ collects labels of $\Lambda'_i \subseteq \Lambda_i$ and produces the configuration $\mathcal{O}^b_i$. One can again assemble these observations by

$$\mathcal{O}^b \quad = \quad \mathcal{O}^b_1 \times_T \ldots \times_T \mathcal{O}^b_N \tag{3.39}$$

which computes all possible interleavings of the sequences $\mathcal{O}^b_i$, and then apply (3.38). The latter will then amount to an unfolding of $\mathcal{A}$ guided by a complex trellis automaton $\mathcal{O}^b$, instead of a sequence. This is feasible, but of course very inefficient, because of the complexity of both $\mathcal{A}$ and $\mathcal{O}^b$. Fig. 3.18 illustrates this point : the two sequences of observations share a single label, β. The result of the product is much more complex than its factors, because it develops all concurrency diamonds. So we are rather looking for diagnosis methods that operate on factorized forms, and avoid expanding large products. This is in fact the heart of the approaches we advocate in this document.

3.6.3 Distributed diagnosis

Before addressing the distributed diagnosis issue, we must introduce the notion of projection of a trellis process on sets of sites $\mathcal{A}_i$.

Projections. The site index set $\{1, \ldots, N\}$ becomes our variable set $\mathcal{V}_{max}$, and we have to define projections Π_I for $I \subseteq \mathcal{V}_{max}$. For simplicity, we will use notation Π_I instead of $\Pi_{\mathcal{A}_I}$, when there is no ambiguity.

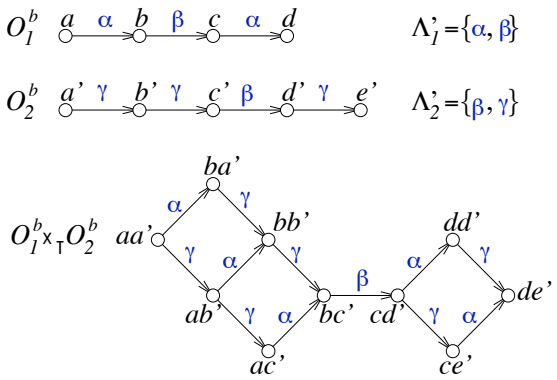


Figure 3.18: *The product of two configurations compute all possible interleavings of these sequences, which results in many concurrency diamonds.*

Let $K, L \subseteq \{1, \ldots, N\}$ be site index sets, and assume $K \subseteq L$. From the product form $\mathcal{U}^{st}(\mathcal{A}_L) = \mathcal{U}^{st}(\mathcal{A}_K) \times_T \mathcal{U}^{st}(\mathcal{A}_{L\setminus K})$ we know there exists a canonical projection

$$\Pi_{L,K} : \mathcal{U}^{st}(\mathcal{A}_L) \to \mathcal{U}^{st}(\mathcal{A}_K) \tag{3.40}$$

Every trellis process of $\mathcal{A}_L$ is a prefix of $\mathcal{U}^{st}(\mathcal{A}_L)$, so $\Pi_{L,K}$ applies to TP of $\mathcal{A}_L$ and yields TP of $\mathcal{A}_K$.

When $K \not\subseteq L$, we define $\Pi_{L,K}$ as $\Pi_{L,K\cap L}$, and since the starting point is generally unambiguous, we simply write Π_K instead of $\Pi_{L,K}$. By associativity of $\times_T$, projections satisfy $\Pi_K \circ \Pi_L = \Pi_{K\cap L}$, which corresponds to axiom (a1).

In practice, projections Π_K are quite easy to implement. Consider for example Fig. 3.18 : it suffices to collapse nodes related by events that do not belong to the sites we want to preserve. The result is almost a trellis process of the selected sites, excepted that it violates the parsimony criterion : isomorphic events may branch out of a given node. Therefore a simple trimming operation is also necessary.

Objective. The central result that we invoke now is the factorization property of the sequential time-unfolding of $\mathcal{A}$. Given $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$, (3.34) becomes

$$\mathcal{U}^{st}(\mathcal{A}) \quad = \quad \mathcal{U}^{st}(\mathcal{A}_1) \times_T \ldots \times_T \mathcal{U}^{st}(\mathcal{A}_N) \tag{3.41}$$

and thus the diagnosis admits the factorized form

$$\mathcal{D} \quad = \quad [\mathcal{U}^{st}(\mathcal{A}_1) \times_T \mathcal{O}_1^b] \times_T \ldots \times_T [\mathcal{U}^{st}(\mathcal{A}_N) \times_T \mathcal{O}_N^b] \tag{3.42}$$

In this equation, each $\mathcal{D}_i = \mathcal{U}^{st}(\mathcal{A}_i) \times_T \mathcal{O}_i^b$ is a *local diagnosis*: it selects runs of site $\mathcal{A}_i$ that explain the local observations $\mathcal{O}_i^b$. The distributed diagnosis problem simply amounts to computing the *minimal product covering of* $\mathcal{D}$, *i.e.* to computing the $\mathcal{T}_i = \Pi_{\mathcal{A}_i}(\mathcal{D})$, which yields

$$\mathcal{D} \quad = \quad [\mathcal{T}_1 \times_T \mathcal{O}_1^b] \times_T \ldots \times_T [\mathcal{T}_N \times_T \mathcal{O}_N^b] \tag{3.43}$$

Equivalently, one could also compute the $\mathcal{D}_i' = \Pi_{\mathcal{A}_i \times \mathcal{O}_i^b}(\mathcal{D}) = \mathcal{T}_i \times_T \mathcal{O}_i^b$ that also satisfy $\mathcal{D} = \mathcal{D}_1' \times_T \ldots \times_T \mathcal{D}_N'$. The latter are called the "reduced" local diagnoses.

In the sequel, we don't distinguish these two reduction problems, that are identical in nature and can be solved in the same manner, with the same complexity.

Apparently, we are thus in a well known land, as explored in chapter 2. In reality, there is a major difference with section 3.4.2, due to the fact that we don't have projections on label sets.

New shape of computations. Our objective is to recover the $\mathcal{D}'_j$ by means of modular computations. Consider two LMCA $\mathcal{A}_1, \mathcal{A}_2$ that share labels $\Lambda_{1,2} \triangleq \Lambda_1 \cap \Lambda_2$, see (3.44). Let $\mathcal{T} = \mathcal{T}_1 \times_T \mathcal{T}_2$ be a trellis process of $\mathcal{A}_1 \times \mathcal{A}_2$, and let us try to replicate the modular computations of section 3.4.2 in order to compute the minimal product covering of $\mathcal{T}$. One would have to project $\mathcal{T}_1$ on $\Lambda_{1,2}$, in order to form a message $\mathcal{M}_{1,2} = \Pi_{\Lambda_{1,2}}(\mathcal{T}_1)$ that would be some sort of label structure. The latter would then be combined by a special product $\times_L$ to $\mathcal{T}_2$ and would produce $\mathcal{T}'_2 = \mathcal{T}_2 \times_L \mathcal{M}_{1,2}$. And symmetrically to obtain $\mathcal{T}'_1$.

$$\mathcal{A}_1 \quad \overset{\Lambda_{1,2}}{\text{———}} \quad \mathcal{A}_2 \tag{3.44}$$

As we have seen, this approach works on simple structures like sets of sequences, but, unfortunately, projections of trellis processes on label sets are hard to define: By simply erasing the events of a trellis process that do not carry a visible label, one would kill the trellis structure. Even worse, this can create circuits, and produce a structure containing fake sequences of visible events[11].

So to transfer information from one side to the other, we bypass this undefined label structure and use a trick, based on more familiar operations: We first combine $\mathcal{T}_1$ and $\mathcal{T}_2$, and project the result on $\mathcal{A}_2$ to form the message: $\mathcal{M}_{1,2} = \Pi_{\mathcal{A}_2}(\mathcal{T}_1 \times_T \mathcal{T}_2)$. One can conclude by $\mathcal{T}'_2 = \mathcal{M}_{1,2} \wedge_T \mathcal{T}_2$. This computation trick looks like a tautology on such a trivial example. But it makes sense to propagate information when one imagines a third site $\mathcal{A}_3$ beyond $\mathcal{A}_2$. As we show below, modular computations are actually based on this idea.

Expression of $\mathcal{A}$ with pullbacks. Since we can't project on labels, we are bound to use projections on sites or sets of sites $\mathcal{A}_i$. So the $\mathcal{A}_i$ become our variables, and to comply with the formalism of chapter 2, we must express our systems in terms of shared variables, *i.e.* in terms of shared sites.

We know that $\mathcal{A}$ can be reformulated as $\mathcal{A} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_M$, where components $\mathcal{S}_j$ are defined by $\mathcal{S}_j = \mathcal{A}_{I_j} = \times_{i \in I_j} \mathcal{A}_i$ and cover all the $\mathcal{A}_i$, *i.e.* $\cup_{1 \le j \le M} I_j = \{1, \ldots, N\}$. By (3.36), one has

$$\mathcal{U}^{st}(\mathcal{A}) \;=\; \mathcal{U}^{st}(\mathcal{S}_1) \wedge_T \ldots \wedge_T \mathcal{U}^{st}(\mathcal{S}_M) \tag{3.45}$$

and similarly

$$\begin{aligned} \mathcal{D} \;&=\; [\mathcal{U}^{st}(\mathcal{S}_1) \times_T \mathcal{O}^b_{I_1}] \wedge_T \ldots \wedge_T [\mathcal{U}^{st}(\mathcal{S}_M) \times_T \mathcal{O}^b_{I_M}] \\ &\triangleq\; \mathcal{D}_1 \wedge_T \ldots \wedge_T \mathcal{D}_M \end{aligned} \tag{3.46}$$

[11] It is not excluded however that more clever height functions would solve this difficulty. But this is still a research topic.

where the local diagnoses $\mathcal{D}_j = \mathcal{U}^{st}(\mathcal{S}_j) \times_T \mathcal{O}^b_{I_j}$ are now computed on components instead of sites. The expression above assumes that every local diagnosis $\mathcal{D}_j$ incorporates by $\mathcal{O}^b_{I_j}$ the observations on all sites $\mathcal{A}_{I_j}$ covered by component $\mathcal{S}_j$. In reality, the duplication of observations $\mathcal{O}^b_i$ is unnecessary : it is sufficient to "distribute" them on components $\mathcal{S}_j$ in such a way that they all appear at least once. However, taking all $\mathcal{O}^b_{I_j}$ to compute $\mathcal{D}_j$ has the advantage to "maximally reduce" $\mathcal{U}^{st}(\mathcal{S}_j)$ before message exchanges start.

The distributed diagnosis problem now amounts to computing the $\mathcal{T}_j = \Pi_{\mathcal{S}_j}(\mathcal{D})$, or equivalently the $\mathcal{D}'_j = \Pi_{\mathcal{S}_j \times \mathcal{O}^b_{I_j}}(\mathcal{D}) = \mathcal{T}_j \times_T \mathcal{O}^b_{I_j}$, that form the minimal "pullback covering" of $\mathcal{D}$:

$$\begin{aligned} \mathcal{D} &= [\mathcal{T}_1 \times_T \mathcal{O}^b_{I_1}] \wedge_T \ldots \wedge_T [\mathcal{T}_N M \times_T \mathcal{O}^b_{I_M}] \\ &= \mathcal{D}'_1 \wedge_T \ldots \wedge_T \mathcal{D}'_M \end{aligned} \tag{3.47}$$

Naturally, the minimal product covering can be obtained by further projecting the $\mathcal{D}'_j$ on the individual sites they represent.

Separation theorem. To relate things to chapter 2 in a clear manner, we now have a variable set $\mathcal{V}_{max} = \{1, \ldots, N\}$, trellis processes $\mathcal{T}_I$ as systems, projections Π_I and the composition operator $\wedge_T$. So we are in the setting of the dual graph representation (section 3.2.2). But at this point, the central axiom (a3) is missing... The reason is that the interactions in $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$ are due to shared labels, and we need to express them in terms of shared *sites*. When one computes $\mathcal{A}_I \wedge \mathcal{A}_K$, the shared sites $\mathcal{A}_{I \cap K}$ do not necessarily capture all interactions between $\mathcal{A}_I$ and $\mathcal{A}_K$: shared labels $\Lambda_I \cap \Lambda_K$ may not all lie within $\Lambda_{I \cap K}$ (see Fig. 3.10 and the discussions in section 3.2.2).

Nevertheless, one has the following property

Theorem 9 *Let $I, K \subseteq \{1, \ldots, N\}$, and take $J = I \cap K$. Let $\mathcal{T}_I, \mathcal{T}_K$ be trellis processes of $\mathcal{A}_I, \mathcal{A}_K$ respectively, and assume that shared sites $\mathcal{A}_J$ capture all interaction labels between $\mathcal{A}_I$ and $\mathcal{A}_K$,* i.e. *$\Lambda_I \cap \Lambda_K \subseteq \Lambda_J$. Then*

$$\Pi_J(\mathcal{T}_I \wedge_T \mathcal{T}_K) = \Pi_J(\mathcal{T}_I) \wedge_T \Pi_J(\mathcal{T}_K) \tag{3.48}$$

This weaker form of axiom (a3) is actually sufficient to derive modular computations. Propagation and merge equations now take the following form :

Proposition 8 *Let $I, J, K \subseteq \{1, \ldots, N\}$, and let $\mathcal{T}_I, \mathcal{T}_J, \mathcal{T}_K$ be trellis processes of $\mathcal{A}_I, \mathcal{A}_J, \mathcal{A}_K$ respectively. If $\Lambda_I \cap \Lambda_K \subseteq \Lambda_J$, denoted $I|J|K$, then*

$$\Pi_J(\mathcal{T}_I \wedge_T \mathcal{T}_J \wedge_T \mathcal{T}_K) = \Pi_J(\mathcal{T}_I \wedge_T \mathcal{T}_J) \wedge_T \Pi_J(\mathcal{T}_J \wedge_T \mathcal{T}_K) \tag{3.49}$$

$$\Pi_I(\mathcal{T}_I \wedge_T \mathcal{T}_J \wedge_T \mathcal{T}_K) = \Pi_I[\mathcal{T}_I \wedge_T \Pi_J(\mathcal{T}_J \wedge_T \mathcal{T}_K)] \tag{3.50}$$

Comparing these equations to the "true" ones (2.8) and (2.10), the reader will notice that proposition 8 simply implements the "trick" described at (3.44).

Support graph of computations. So far, no choice was made to decompose $\mathcal{A}$ into a pullback of components and obtain (3.45). We now exploit this degree of freedom, that was already discussed in section 3.2.2, dedicated to dual graph representations.

Let us define components $\mathcal{S}_k$ in such a way that they cover a communication graph $\mathcal{G}^c$ between sites $\mathcal{A}_i$ (Fig. 3.19). Then form a communication graph $\mathcal{G}^{c\prime}$ between these components. The latter forms the support of computations. Indeed, on graph $\mathcal{G}^{c\prime}$, the separation property holds, which means that theorem 9 applies, so propagations and merges of proposition 8 are legal and the MPA leads us to the desired reduced components $\mathcal{S}'_j$.

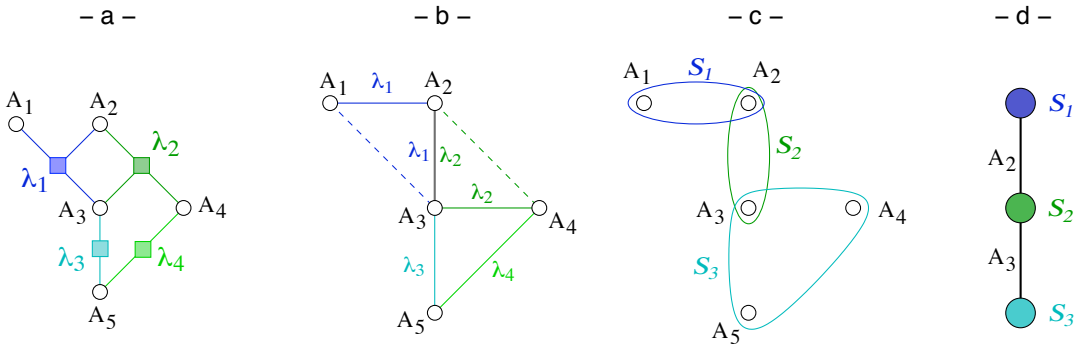


Figure 3.19: *From left to right: label constraints between sites $\mathcal{A}_i$, a communication graph $\mathcal{G}^c$ of the $\mathcal{A}_i$, a definition of components $\mathcal{S}_j$ covering $\mathcal{G}^c$, and a connectivity graph between components $\mathcal{S}_j$.*

We didn't mention sites corresponding to measurements, *i.e.* the $\mathcal{O}_i$. They actually play no role in the structure of interactions, even if the $\mathcal{O}_i$ observe shared labels. Indeed, each $\mathcal{O}_i$ is separated from all other sites (including measurement sites) by site $\mathcal{A}_i$. So introducing the $\mathcal{O}_i$ in Fig. 3.19.b simply amounts to putting an edge between each $\mathcal{A}_i$ and the associated $\mathcal{O}_i$; all edges between the $\mathcal{O}_i$, if any, are dotted (*i.e.* redundant). Moreover, since the $\mathcal{O}_i$ correspond to observations collected on a true run of $\mathcal{A}$, the projections $\Pi_{\mathcal{O}_i}(\mathcal{D})$ necessarily yield $\mathcal{O}_i$. In summary, measurements can just be ignored in the graphical representations.

Involutivity. It is not hard to check that trellis processes form involutive systems: for $\mathcal{T}_I$ a TP of $\mathcal{A}_I$, one has $\mathcal{T}_I \wedge_T \Pi_J(\mathcal{T}_I) = \mathcal{T}_I$. So we are in the nice situation where turbo procedures converge, if the support graph of computations is not a tree. The progressive reduction performed by MPA (theorem 4) and the local extendibility property of the limit (theorem 3) are thus granted.

3.7 Towards true concurrency semantics

This section comes back to the multi-clock aspects, in order to (try to) convince the reader of its necessity for modular computations.

Consider the two LMCA $\mathcal{A}_1, \mathcal{A}_2$ of Fig. 3.20. They synchronize on labels $\{\alpha, \beta, \gamma\}$,

and transitions t_5 and t'_4 are private to $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. Fig. 3.21 depicts two trellis processes $\mathcal{T}_1, \mathcal{T}_2$ of these LMCA, together with their "products" for different choices of height functions. The first product (3rd TP on the figure) corresponds to $\mathcal{T}_1 \times_T \mathcal{T}_2$ in the multi-clock setting : time is counted independently in each component, which prevents the merge of the two conditions labeled (c, c'), since they correspond to different values of the vector clock.

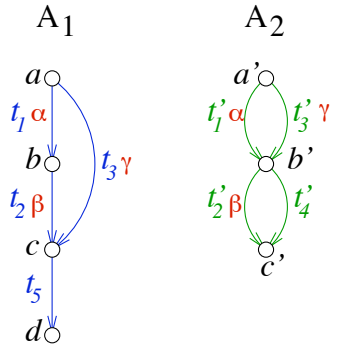


Figure 3.20: *Two labeled multi-clock automata, sharing labels* $\{\alpha, \beta, \gamma\}$.

Assume that one would decide to abandon vector clocks and choose to use a global counting of time in the definition of trellis processes. The "product" $\mathcal{T}_1 \times_T \mathcal{T}_2$ would then result in the rightmost TP of Fig. 3.21 where the two conditions labeled (c, c') are now merged, since they both have two events in their past. Projecting back the result on $\mathcal{A}_1$ (by erasing events of $\mathcal{A}_2$) underlines the bad properties of this construction : this projection yields a "trellis process" isomorphic to $\mathcal{A}_1$. So the result is not any more a correctly folded trellis process of $\mathcal{A}_1$. But worse than that, this projection introduces the extra run $a[t_3\rangle c[t_5\rangle d$ that was not present initially in the factor $\mathcal{T}_1$. So there is little hope that modular computations based on this notion of trellis process would produce correct results.

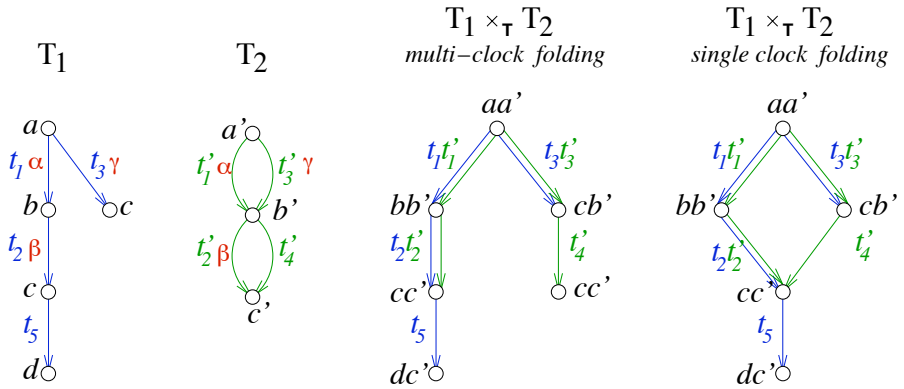


Figure 3.21: *Two trellis processes of the previous LMCA, their correct product, and a product based on a global notion of time.*

To explain the bad properties of this "single-clock product" $\mathcal{T} = \mathcal{T}_1 \times_T^{sc} \mathcal{T}_2$ by the more formal aspects of our construction, notice that $\mathcal{T}$ doesn't correspond to

a categorical product. There doesn't exist morphisms from $\mathcal{T}$ to its two factors $\mathcal{T}_1$ and $\mathcal{T}_2$ (actually there is no morphism to $\mathcal{T}_1$). In other words, considering $\mathcal{T}$ as a valid trellis process would violate the universal property stated in theorem 8 on $\mathcal{T}_1$. There exists a morphism from $\mathcal{T}$ to $\mathcal{A}_1$, that doesn't decomposes into a morphism to $\mathcal{T}_1$ followed by the folding of $\mathcal{T}_1$ into $\mathcal{A}_1$.

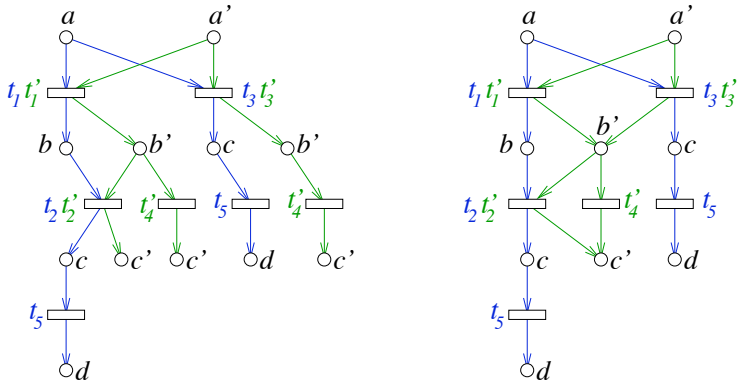


Figure 3.22: *Unfolding and time-unfolding of $\mathcal{A}_1 \times \mathcal{A}_2$ of Fig. 3.20.*

Since one is bound to the use of vector clocks, this means that the relevant notion of time for distributed systems is itself distributed [84, 50]. Rather than a universal linear time, one has local notions of time, plus synchronizations on special events. Pushing further this idea, it makes sense to develop a setting where transitions themselves would have local effects. This suggests to use a distributed notion of state, rather than global states, and adopt true concurrency semantics.

Consider for example $\mathcal{A}_1$ and $\mathcal{A}_2$ of Fig. 3.20. These new semantics will lead to a new notion of product for $\mathcal{A}_1 \times \mathcal{A}_2$, as well as new notions of unfolding $\mathcal{U}(\mathcal{A}_1 \times \mathcal{A}_2)$ (Fig. 3.22 left) and of time-unfolding $\mathcal{U}^t(\mathcal{A}_1 \times \mathcal{A}_2)$ (Fig. 3.22, right). On this example, the product $A_1 \times \mathcal{A}_2$ would actually look like the latter : simply merge the subnets below the two conditions labeled c.

These new representations of sets of runs, $\mathcal{U}(\mathcal{A})$ and $\mathcal{U}^t(\mathcal{A})$, display explicitly the concurrency of events : for example the concurrent firings of t_4 and t_5. In this setting, the product of two systems that have no interaction will amount to their juxtaposition. That will avoid the useless computation of concurrency diamonds, and result in even more compact data structures to encode sets of runs. The next chapter examines computations based on unfoldings, in the true concurrency semantics, and chapter 5 does the same with trellises, or time-unfoldings.

As an extra motivation to move to true concurrency semantics, let us illustrate a last drawback of sequential semantics. We have seen that vector clocks are necessary to modular computations. Nevertheless, they have very unpleasant side effects : in some cases, there may exist a drift between clocks of two components. In particular when these components have concurrent behaviors (*i.e.* local transitions). In terms of trellis processes, this results in very large structures, as illustrated in Fig. 3.23. These phenomena suggest not to recommend the approach of this chapter, and use instead true concurrency semantics, that will erase such weird behaviors.

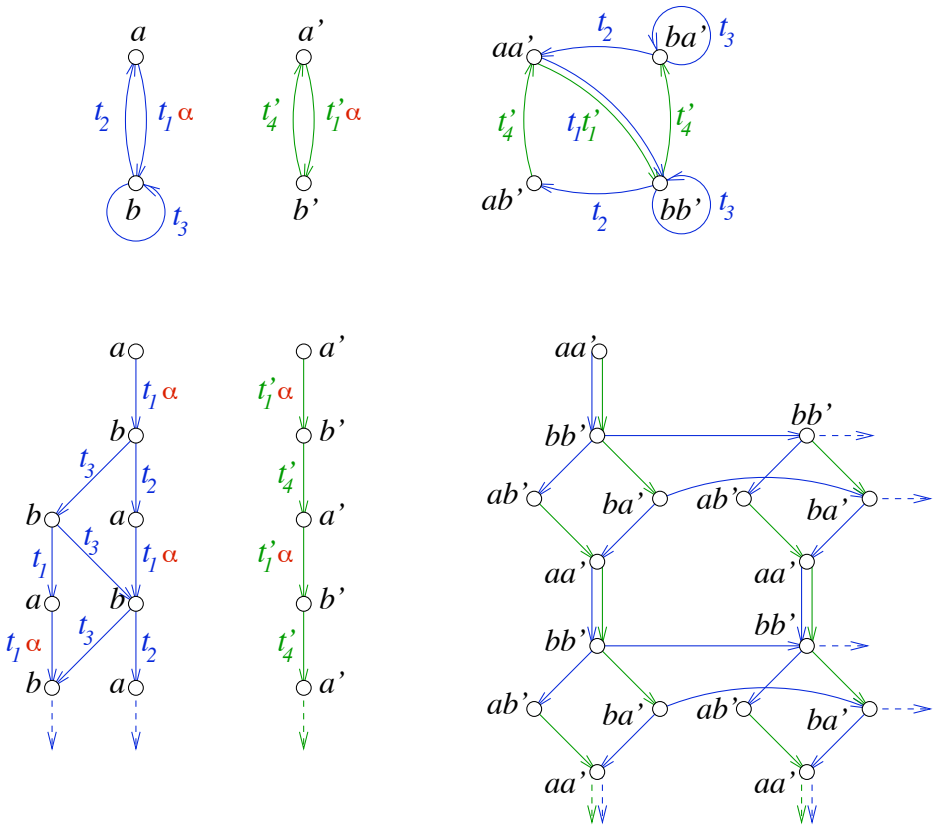


Figure 3.23: *Top : two LMCA* $\mathcal{A}_1, \mathcal{A}_2$ *that share label* α*, and their product* $\mathcal{A}_1 \times \mathcal{A}_2$*. Bottom : the STU* $\mathcal{U}^{st}(\mathcal{A}_1), \mathcal{U}^{st}(\mathcal{A}_2)$ *and their product* $\mathcal{U}^{st}(\mathcal{A}_1 \times \mathcal{A}_2)$*.*

3.8 Summary

We have proposed a methodology to deal with distributed systems by means of distributed or modular algorithms, with the diagnosis problem as guiding application. To do so, we first introduced the notion of network of automata. Mathematically, one can easily derive modular diagnosis algorithms based on component languages : they take the form of message passing algorithms, originally derived for Bayesian networks. The keystone of the construction is a factorization property on sets of runs of the global system, just like in Bayesian networks the factorization property of the global likelihood function is the key to fast algorithms.

Nevertheless, computing with component languages demands important storage capabilities, which is not suited to practical applications, in particular if we aim at on-line monitoring algorithms. So we have proposed a more compact way of encoding sets of runs, by means of trellis processes. A well known notion in different research communities, that has been adapted here to networks of automata. This adaptation required to introduce the concept of multi-clock, *i.e.* to keep track of components when they are assembled into networks. Without this concept, it seems difficult to obtain the factorization property on trellis processes of a compound system, which is essential to modular computations. As soon as this property is derived, which is most easily done with category theory arguments, the mathematical framework developed for languages applies to trellis processes, up to some technical modifications.

Introducing vector clocks in distributed systems is a first step to a correct modeling of concurrent behaviors. This is what we do in the next chapters, moving from sequential semantics to true concurrency semantics. We will show that the mathematical skeleton of this chapter still applies, up to some technical extensions, and that we can obtain even more compact descriptions of sets of runs.

Related work

The diagnoser approach has been first investigated in [95], with a focus on diagnosability issues. Diagnosability means that a bounded number of observations after the occurrence of a fault is sufficient to detect the failure. This amounts to checking the absence of ambiguous circuits in the diagnoser, *i.e.* (in the formalism of section 3.3.3) circuits of the observer of $\mathcal{A} \times \mathcal{B}$ that contain state estimates of the two kinds, green and red. Issues appearing with distributed observations and/or systems have been soon considered. The protocol-based approach of [25], for example, assumes distributed observations on separate components, but global diagnosers are used at each sensor, and the focus is on the information they should exchange to merge their estimates. By contrast, [21, 22] introduce a true distributed approach, where local diagnosers are based on a local model. Similar results are also derived in [54, 55], with a slightly different model : components are defined as Petri nets instead of automata. In both cases, component interactions are assumed to be visible, which guarantees that the diagnoser/observer of the global model has a product form, and thus enables modular solutions. Unfortunately, the problem is not so much analyzed from this angle, which hides the important phenomena that make things work. For example the fact that, when interactions are observed, no

matter which interleaving of the observed sequences is chosen in Fig. 3.18, the same estimated states are obtained.

The literature related to distributed diagnosis is vast and diverse. Several approaches make use of message passing (or belief propagation) algorithms, in relation to the topology of the supervised system, but assume no dynamics in the components. For example [89] deploys MPA on failure trees, and [97, 98] use dependency graphs between the elementary functions that compose a network. All these models are stochastic but static.

As already mentioned, the closest contributions to chapter 3 are due to Su and Wonham [102, 103]. They do use networks of dynamic systems, where each component is specified as a language. The approach is purely algebraic : no randomization is assumed, by contrast with the previous ones. Let us also mention the contributions of Pencole [88], or Baroni *et al.* [8]. They both model a distributed system as a network of communicating automata, and consider the distributed on-line construction of a "diagnoser" (*sic*), *i.e.* a system whose runs are exactly the explanations to the distributed observations. Although the idea is to progressively assemble the "local diagnosers," by a product, this is not properly a message passing solution : there is no notion of projection. But strategies are deployed in [88] to avoid building useless parts in the recursive composition of local diagnosers. Interestingly, the idea of not unfolding silent loops is present in these two contributions.

Finally, let us mention the very original chronicle approach [28, 30, 49]. By contrast with the previous ones, it doesn't look for an explanation to *all* alarms or symptoms. It rather operates as a collection of filters that extract relevant patterns of observations in a sequence. These patterns are specified as partial orders of labels, related by time constraints. There exist both fast chronicle recognition algorithms, and learning algorithms. Another difference with model-based approaches is that the expert knowledge must come afterwards, to evaluate the relevance of learned chronicles and to associate them to a failure "diagnosis." There exist techniques, however, to merge chronicles with topology models [5, 56].

Chapter 4

True concurrency semantics

So far, we have defined a distributed system as a network of components, with localized interactions. We have shown that some monitoring problems, like (maximum likelihood) trajectory estimation, that we called the diagnosis problem, could be solved in a distributed manner. The modular/distributed resolution method exploits the factorization property of sets of runs, under the various forms where these runs can be described. In the sequential semantics, run sets can be encoded either

- as sets of sequences (of labels or transitions),
- as branching processes or decision trees, or
- as trellis processes,

which establishes a hierarchy in terms of compactness. Moving to true concurrency semantics, this hierarchy remains. It is also remarkable that the algebraic structure of distributed computations is preserved. Extra gains are obtained in terms of compactness of the objects we handle, at the price of a little more technicality in the proof of the key results.

This chapter presents the interest of branching processes, and the new features they introduce in the distributed diagnosis problem. Chapter 5 will make one more step and study the counterpart of trellis processes.

4.1 Networks of automata as asynchronous systems

The previous chapter introduced a notion of component, somehow artificially, into the ordinary definition of an automaton: in $\mathcal{A} = (S, T, \to, s^0, \chi, I, \lambda, \Lambda)$, I is an index set that gives names to components, and the function $\chi : T \to 2^I$ defines on which components every transition $t \in T$ "operates." Although components are formally added by the product of (labeled) multi-clock automata, the composition nevertheless results in an automaton. Fig. 4.1 illustrates this idea, in a Petri-net-like representation of automata.

This formalism suffers from several drawbacks, some of which have been illustrated at the end of the previous chapter. Let us underline two of them.

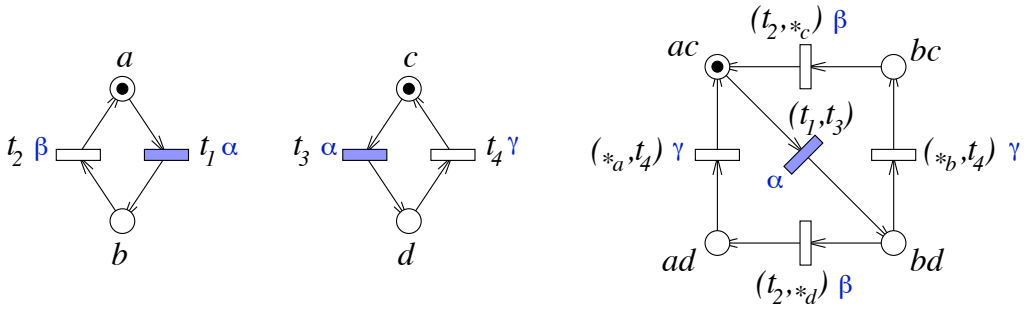


Figure 4.1: *Two LMCA (left) that share only label α, and their product (right).*

Numbers of states and transitions are multiplied by the composition, and in particular each private transition of one factor is duplicated n times, where n is then number of states of the other factor. So we explicitly represent the fact that one LMCA is waiting when the other performs a private transition (see the duplications of t_2 and t_4 in the example).

Secondly, runs of a product LMCA $\mathcal{A}_1 \times \mathcal{A}_2$ are almost bound to be sequences of events : the fact that the reset transitions t_2 and t_4 are concurrent is not exploited. So one distinguishes $(t_1, t_3)(t_2, \star_d)(\star_a, t_4)$ from $(t_1, t_3)(\star_b, t_4)(t_2, \star_c)$, whereas it would be preferable to write $(t_1, t_3)t_2t_4$ and $(t_1, t_3)t_4t_2$, and even consider these two runs as equivalent, since t_2 and t_4 can be permuted without altering the final state of $\mathcal{A}_1 \times \mathcal{A}_2$.

The objective of this chapter is precisely to avoid the redundancy introduced by the sequential semantics, and explicitly take advantage of the concurrency between events. We'll take as definition of a run not a sequence of transitions, but an equivalence class of sequences, where equivalence means "identical up to the permutation of two successive concurrent events." In the example above, $\{\, (t_1, t_3)t_2t_4\,,\, (t_1, t_3)t_4t_2\,\}$ will be considered as a single run. In other words, sequences will be replaced by Mazurkiewicz *traces* [27].

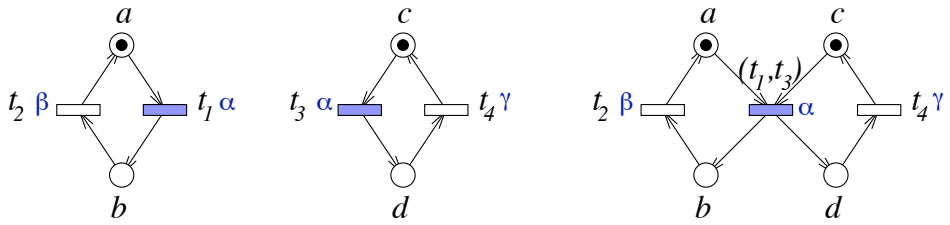


Figure 4.2: *Two labeled automata (left) combined into a safe Petri net (right).*

To do so, a natural step consists in replacing automata by asynchronous systems. This is simply obtained by changing the notion of composition : in substance, we take the disjoint union of states (instead of the product), and glue transitions that carry identical labels, in all possible ways, to form joint transitions. Private transitions remain unchanged. As illustrated in Fig. 4.2, this yields a Petri net : we now have several "tokens" to identify a global state of the system. Firing (t_1, t_3) moves simultaneously the two tokens to b and d, after what they can return to their original

places independently. Observe that the number of tokens remains constant : this illustrates the fact that we now have two components, or equivalently two (local) state variables, instead of one.

There exist several formalisms to define and handle such systems. Our original contributions [37, 40] chose to emphasize the presence of several state variable, which lead to the notion of tile system (see also the introduction, section 1.3). Alternatively, one could simply consider safe Petri nets. But since we need to preserve the notion of component, or of variable, the latter must be slightly enriched. We therefore introduce the notion of multi-clock net (MCN).

This chapter is organized as the previous one : we first define the category of (labeled) multi-clock nets, and their composition. We then provide them with a notion of trajectory, in the true concurrency semantics. As for multi-clock automata, a crucial problem is the compact representation of sets of trajectories. We focus here on the notions of branching process and unfolding of a MCN, the natural counterparts of the sequential trellis process and the sequential time-unfolding, excepted that merges are not allowed. So these objects are more on the side of decision trees. It is shown that one can actually compute with them, pretty much like with sequential trellis processes. But specific difficulties of the true concurrency semantics blur the nice algebraic setting of the previous chapter. We show that computations must actually be performed with more complex objects, that we call augmented branching processes.

4.2 Multi-clock nets and their composition

4.2.1 A category of multi-clock nets

Petri net. An *ordinary Petri net* is a 4-tuple $\mathcal{N} = (P, T, \rightarrow, M^0)$ where P, T are respectively the place and transition sets[1], and $\rightarrow \subseteq (P \times T) \cup (T \times P)$ is the flow relation relating places and transitions[2]. $M^0 : P \rightarrow \mathbb{N}$ is a finite multi-set on P representing the initial *marking* of the net, *i.e.* the number of tokens that each place initially holds. Given a node $x \in P \cup T$, ${}^\bullet x$ and $x^\bullet$ stand for pre- and post-sets, *i.e.* the immediate predecessors and successors of x in $\rightarrow$. By contrast with the previous chapter, these sets are not any more constrained to be singletons.

A transition $t \in T$ is *enabled* (or *activated*) in a marking M, denoted by $M[t\rangle$, iff $\forall p \in {}^\bullet t, M(p) > 0$. Firing t from M yields the new marking $M' = M - {}^\bullet t + t^\bullet$, which we denote by $M[t\rangle M'$. M is a reachable marking iff there exists a sequence $\sigma = (t_1, t_2, \ldots, t_n)$ of transitions (or a *run*) and intermediate markings such that $M^0[t_1\rangle M_1[t_2\rangle M_2 \ldots [t_N\rangle M_N = M$, abbreviated into $M^0[\sigma\rangle M$. We limit ourselves to *safe* nets, *i.e.* nets for which places hold at most one token in any reachable marking. We thus identify markings to subsets of P, and M^0 to $P^0 \subset P$.

Multi-clock net. A *multi-clock net* (MCN), or *net* for short, is a tuple $\mathcal{N} = (P, T, \rightarrow, P^0, \nu)$ where

[1] We do not require that P and T be finite.

[2] We assume each place is related to one transition at least, and each transition has at least one input place and one output place.

1. $(P, T, \rightarrow, P^0)$ is an ordinary safe net,

2. $\nu : P \rightarrow P^0$ defines a partition on places, and the restriction $\nu_{|P^0}$ is the identity ; we denote by $\bar{p}$ the equivalence class $\nu^{-1}(\nu(p))$ of a place p,

3. $\forall t \in T$, ν is injective on ${}^\bullet t$ and on $t^\bullet$, and $\nu({}^\bullet t) = \nu(t^\bullet)$.

This definition deserves some comments.

Observe first that every transition satisfies $|{}^\bullet t| = |t^\bullet|$. So the number of tokens remains constant in a MCN. Moreover, let $M \subseteq P$ be a reachable marking of $\mathcal{N}$, one has $\nu_{|M}$ is bijective. In other words, let $p \in P^0$, at any time there is exactly one place in $\nu^{-1}(p)$ holding a token[3].

Secondly, consider $\mathcal{N}_{|\bar{p}}$, the restriction of $\mathcal{N}$ to places of $\bar{p}$, $p \in P$. $\mathcal{N}_{|\bar{p}}$ is an automaton, *i.e.* a Petri net where a single place holds a token at any time, as in Fig. 4.1. Therefore, a multi-clock net can be regarded as a synchronous product of automata, as illustrated in Fig. 4.2.

Relating this formalism to the previous chapter, P^0 now replaces the index set I to name components, and the χ function on transitions is replaced by ν on places. So a transition t operates on components $\nu({}^\bullet t) = \nu(t^\bullet)$ instead of $\chi(t)$. By abuse of vocabulary, we will sometimes consider $\bar{p}$ as the state variable of component $\mathcal{N}_{|\bar{p}}$ (more rigorously, it corresponds to the set of possible values of this state variable).

Naturally, a *labeled multi-clock net* (LMCN) is a MCN enriched with a labeling function on transitions : $\mathcal{N} = (P, T, \rightarrow, P^0, \nu, \lambda, \Lambda)$, with $\lambda : T \rightarrow \Lambda$.

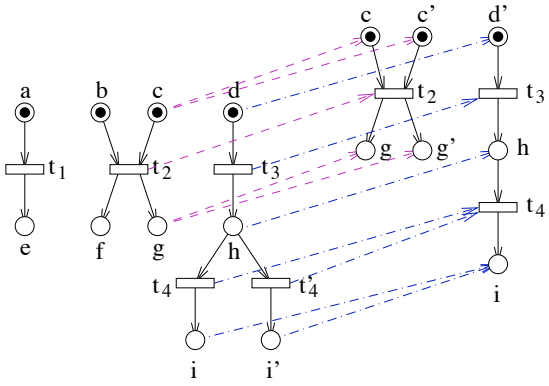


Figure 4.3: *A typical morphism between two multi-clock nets, indicated by dashed arrows. In the left MCN, ν_1 defines the partition $\{a, e\}$, $\{b, f\}$, $\{c, g\}$, $\{d, h, i, i'\}$ on places.*

Morphism. To turn the collection of multi-clock nets into a category, we need the extra notion of morphism between nets. Let $\mathcal{N}_1, \mathcal{N}_2$ be two MCN, with $\mathcal{N}_i = (P_i, T_i, \rightarrow_i, P_i^0, \nu_i)$, $i = 1, 2$. A *morphism* ϕ from $\mathcal{N}_1$ to $\mathcal{N}_2$ is defined as a pair (ϕ_P, ϕ_T) where

[3]Notice that it is always possible to turn a safe net $\mathcal{N}$ into a multi-clock net with essentially the same behavior, simply by adding to each place of $\mathcal{N}$ a complementary place. So multi-clock nets are almost equivalent to safe nets.

1. ϕ_T is a partial function from T_1 to T_2, and ϕ_P a relation between P_1 and P_2,
2. $P_2^0 = \phi_P(P_1^0)$ and $\phi_P^{op} : P_2^0 \to P_1^0$ is a total function, where ϕ_P^{op} denotes the opposite relation to ϕ_P,
3. if $p_1 \, \phi_P \, p_2$ then the restrictions $\phi_T : {}^\bullet p_1 \to {}^\bullet p_2$ and $\phi_T : p_1^\bullet \to p_2^\bullet$ are total functions,
4. if $t_2 = \phi_T(t_1)$ then the restrictions $\phi_P^{op} : {}^\bullet t_2 \to {}^\bullet t_1$ and $\phi_P^{op} : t_2^\bullet \to t_1^\bullet$ are total functions,
5. ϕ_P preserves the partitioning of places: $\forall (p_1, p_2) \in P_1 \times P_2, \quad p_1 \, \phi_P \, p_2 \;\Rightarrow\; \nu_1(p_1) \, \phi_P \, \nu_2(p_2)$

Notice that points 3 and 4 entail that the pair (ϕ_P, ϕ_T) preserves the flow relation (on its domain of definition), and so morphisms will preserve runs, which is the first property one expects from them. It may look surprising that ϕ_P is a relation and not a partial function. This generalization allows ϕ_P to duplicate places, which is necessary to the existence of a categorical product (recall that this duplication ability was also present in MCA morphisms). In conjunction with the last requirement, one can actually show that

Lemma 7 *Let $\phi : \mathcal{N}_1 \to \mathcal{N}_2$ be a morphism of MCNs, such that every automaton $N_{i|\bar{p}}$ has a single connected component, then ϕ erases or duplicates state variables (or components) as a whole, just like MCA morphisms (see Fig. 4.3). Specifically, one has*

a. the inverse image by ϕ of a class of ν_2 is included in a class of ν_1 ;

b. given a class of ν_1, ϕ is either defined on all elements of this class, or on none of them ;

c. when a place $p_1 \in P_1$ is duplicated by ϕ, i.e. *related to (elements of) several classes of ν_2, each place in $\bar{p_1}$ is duplicated in the same way,* i.e. *related to the same classes.*

Finally, for labeled MCN $\mathcal{N}_i = (P_i, T_i, \to_i, P_i^0, \nu_i, \lambda_i, \Lambda_i), i = 1, 2$, the definition of a morphism $\phi : \mathcal{N}_1 \to \mathcal{N}_2$ must be reinforced by extra requirements (that rephrase those of labeled MCA) :

6. $\Lambda_2 \subseteq \Lambda_1$, *i.e.* ϕ reduces the label set,
7. $Dom(\phi_T) = \lambda_1^{-1}(\Lambda_2)$, *i.e.* ϕ is defined on transitions carrying a shared label, and only on them,
8. if $\phi_T(t_1) = t_2 \neq \star$ then $\lambda_1(t_1) = \lambda_2(t_2)$, *i.e.* ϕ preserves labels on its domain of definition $Dom(\phi_T)$.

We denote by *Nets* the category having the LMC nets as objects, and the above morphisms as arrows. By abuse of notations, we simply write ϕ instead of ϕ_S or ϕ_T, and $\phi(X)$ to denote places in relation with at least one place in X.

4.2.2 Composition by product and pullback

Product. Let $\mathcal{N}_1, \mathcal{N}_2$ be two LMCN, their product $\mathcal{N}_1 \times \mathcal{N}_2$ is defined as the triple $(\mathcal{N}, \psi_1, \psi_2)$ where $\mathcal{N} = (P, T, \to, P^0, \nu, \lambda, \Lambda)$ is a net and $\psi_i : \mathcal{N} \to \mathcal{N}_i$ a morphism, such that

1. $P = \{(p_1, \star) : p_1 \in P_1\} \cup \{(\star, p_2) : p_2 \in P_2\}$;
   $\psi_1(p_1, \star) = p_1$ and $\psi_1(\star, p_2) = \star$ (*i.e.* undefined), and symmetrically for ψ_2,
2. $P^0 = \psi_1^{-1}(P_1^0) \cup \psi_2^{-1}(P_2^0)$,
3. $T = T_s \cup T_p$ where

$$T_s = \{(t_1, t_2) \in T_1 \times T_2 \ : \lambda_1(t_1) = \lambda_2(t_2)\} \tag{4.1}$$

$$T_p = \{(t_1, \star) \ : t_1 \in T_1,\ \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} \cup \{(\star, t_2) \ : t_2 \in T_2,\ \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} \tag{4.2}$$

   and $\psi_i(t_1, t_2) = t_i$ if $t_i \neq \star$ and is undefined otherwise,
4. $\to$ is defined by ${}^\bullet(t_1, t_2) = \psi_1^{-1}({}^\bullet t_1) \cup \psi_2^{-1}({}^\bullet t_2)$, assuming ${}^\bullet\star = \emptyset$, and similarly for $(t_1, t_2)^\bullet$,
5. $\Lambda = \Lambda_1 \cup \Lambda_2$ and λ follows accordingly,
6. ν is simply the union of partitions ν_1 and ν_2.

In a product, each component preserves its places by the disjoint union in (1), and components are added (6). As for LMCA, transitions carrying a shared label synchronize (4.1), provided they find a partner, while those carrying a private label remain unchanged (4.2). Notice that private transitions are not duplicated, by contrast with LMCA. See Fig. 4.4 for a simple example.

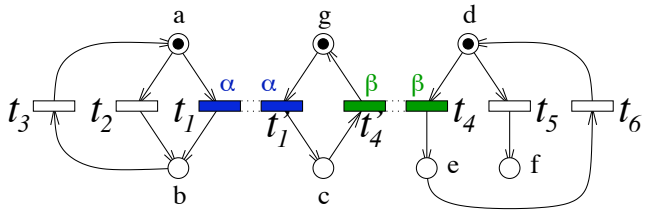


Figure 4.4: *Product of three nets, each one with a single component. On this simple example, the product amounts to gluing transitions with identical labels.*

Proposition 9 $\times$ *is the categorical product in Nets* , i.e. *the universal property stated in proposition 5 for LMC automata holds also for LMC nets.*

See [112] for a proof in the unlabeled case, which easily specializes.

Pullback. The composition by pullback, *i.e. via* a shared component, can also be defined for LMC nets, and it enjoys the same properties as for LMC automata. The general definition is a little complex [43], so we focus on the more intuitive case where $\mathcal{N}_1, \mathcal{N}_2$ are related to their interface $\mathcal{N}_0$ by partial functions $f_i : \mathcal{N}_i \to \mathcal{N}_0$ instead of general morphisms. The construction resembles very much the one of the product : it actually coincides with it outside the domains of f_1 and f_2. $\mathcal{N}$ and the morphisms $\psi_i : \mathcal{N} \to \mathcal{N}_i$ are given by

1. $P = P_c \cup P_p$ where

$$\begin{aligned} P_c &= \{(p_1, p_2) : f_1(p_1) = f_2(p_2) = p_0 \in P_0\} && (4.3) \\ P_p &= \{(p_1, \star) : f_1(p_1) = \star\} \cup \{ (\star, p_2) : f_2(p_2) = \star\} && (4.4) \end{aligned}$$

   $\psi_i(p_1, p_2) = p_i$ if $p_i \neq \star$ and undefined otherwise,

2. $P^0 = \psi_1^{-1}(P_1^0) \cup \psi_2^{-1}(P_2^0)$,

3. $T = T_c \cup T_s \cup T_p$ where

$$\begin{aligned} T_c &= \{(t_1, t_2) : f_1(t_1) = f_2(t_2) = t_0 \in T_0\} && (4.5) \\ T_s &= \{(t_1, t_2) : f_1(t_1) = \star = f_2(t_2),\ \lambda_1(t_1) = \lambda_2(t_2)\} && (4.6) \\ T_p &= \{(t_1, \star) : f_1(t_1) = \star,\ \lambda_1(t_1) \in \Lambda_1 \setminus \Lambda_2\} \\ &\quad \cup \{(\star, t_2) : f_2(t_2) = \star,\ \lambda_2(t_2) \in \Lambda_2 \setminus \Lambda_1\} && (4.7) \end{aligned}$$

   and $\psi_i(t_1, t_2) = t_i$ if $t_i \neq \star$ and is undefined otherwise,

4. $\to$ is defined by ${}^\bullet(t_1, t_2) = \psi_1^{-1}({}^\bullet t_1) \cup \psi_2^{-1}({}^\bullet t_2)$, assuming ${}^\bullet\star = \emptyset$, and similarly for $(t_1, t_2)^\bullet$,

5. $\Lambda = \Lambda_1 \cup \Lambda_2$ and λ follows accordingly,

6. ν is the union of partitions ν_1 and ν_2 : they coincide on the common places of $\psi_i(P_c)$.

The novelty appears in the special treatment of common places (4.3) and common transitions (4.5).

Proposition 10 *The composition by $\wedge$ corresponds to a pullback in the category Nets .*

The proof of the universal property of the pullback, see (3.3) and Fig. 3.4, can be found in [43]. So we are on solid ground, and derivations of the previous chapter based on category theory arguments still hold : we only changed objects. In particular, the relations between $\times$ and $\wedge$ expressed in proposition 6 remain valid for LMC nets, as illustrated in Fig. 4.5.

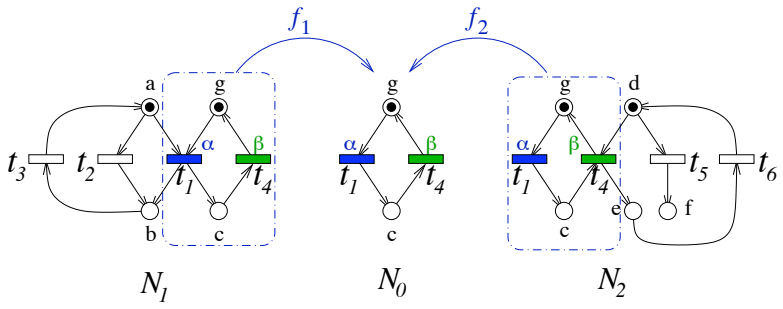


Figure 4.5: *Two nets, that share a common component. The limit of this diagram (pullback) results in the net of Fig. 4.4.*

4.2.3 Graphs associated to a multi-clock system

The situation is exactly as for labeled multi-clock automata (section 3.2). We start from a net $\mathcal{N}$ defined as a product $\mathcal{N} = \mathcal{N}_1 \times \ldots \times \mathcal{N}_N$, and want to represent graphically the internal interactions of $\mathcal{N}$. The objective is of course to prepare the ground for distributed/modular algorithms.

There are essentially two ways to do so : either by a direct graph representation (the common resources, or variables, are the shared labels) or by a dual graph representation (the common resources are the *sites* $\mathcal{N}_i$). Here, we will mostly use the latter : the choice is indeed governed by the type of projections we use in the sequel, either projections on label sets, or projections on sites. It is not excluded that the branching processes we introduce below for modular computations could admit projections on label sets ; we are actually working on the topic. But so far this theory is not available, and we can only project on (sets of) sites. This choice thus imposes to base computations on pullbacks, as in section 3.6.3.

In summary, thanks to proposition 6 and property (3.8) expressed on nets, we can decompose $\mathcal{N}$ as

$$\mathcal{N} = \bigwedge_{j \in J} S_j \qquad \text{with} \qquad \mathcal{S}_j = \times_{i \in I_j} \mathcal{N}_i \tag{4.8}$$

assuming *components* $\mathcal{S}_j$ cover a communication graph between sites $\mathcal{N}_i$ (Figs. 3.9-e and 3.9-f).

4.3 Trajectory sets in the true concurrency semantics

This section defines the notion of trajectory of a net, in the true concurrency semantics, and proposes means to represent sets of runs and compute with them. In the previous chapter, a set of runs of an LMCA could be understood and handled in four different manners : as a set of label sequences (sub-language), a set of transition sequences (a richer notion of sub-language), and more compactly as a branching process, or as a trellis processes. The same ideas would apply here in the true concurrency semantics. We skip the first one, limited to labels, and directly move to runs expressed as *traces* on transitions, that we will represent as *configurations*. And

since configurations will appear as specific cases of branching processes, we directly move to the latter and show how to compute with these objects. Trellis processes form the body of the next chapter.

4.3.1 Unfolding of a net

Occurrence net. The LMC net $\mathcal{O} = (C, E, \rightarrow, C^0, \nu, \lambda, \Lambda)$ is a *(labeled multi-clock) occurrence net* (LMCON, or ON for short) iff it satisfies:

1. $C^0 = \{c \in C : {}^\bullet c = \emptyset\}$,
2. the *causality* relation $\rightarrow^*$, irreflexive transitive closure of $\rightarrow$, is a partial order, and $\forall x \in C \cup E,\ [x] \triangleq \{x\} \cup \{y \in C \cup E : y \rightarrow^* x\}$ is finite,
3. $\forall c \in C,\ |{}^\bullet c| \leq 1$,
4. the *conflict* relation $\#$ defined by the two properties below is irreflexive :
   (a) $\forall e, e' \in E,\quad [e \neq e',\ {}^\bullet e \cap {}^\bullet e' \neq \emptyset] \ \Rightarrow\ e \# e'$,
   (b) $\forall x, x' \in C \cup E,\quad [\exists e, e' \in E,\ e \# e',\ e \rightarrow^* x,\ e' \rightarrow^* x'] \ \Rightarrow\ x \# x'$.

The change in notations accounts for the usual terminology of *conditions*, instead of places, and *events*, instead of transitions. In an LMCON, "time" is unfolded, as indicated by (2). A condition can be marked by a unique event (3). By contrast, it may enable several events, which corresponds to a conflict situation. This creates a branching in the net, and the corresponding branches will never meet each other again (4). So conflicts are also unfolded. See Fig. 4.6, center, for an example.

Configuration. LMCONs are generally introduced to represent runs of a net in the so-called *true concurrency semantics*. To do so, we need extra elements of terminology about ONs. Two nodes $x, x' \in C \cup E$ are said to be *concurrent*, denoted by $x \perp\!\!\!\perp x'$, iff neither $x \# x'$ nor $x \rightarrow^* x'$ nor $x' \rightarrow^* x$ holds. A *co-set* is a set of pairwise concurrent conditions, and a *cut* is a maximal co-set for the inclusion. Finally, a *configuration* is a sub-net κ of $C \cup E$ which is conflict-free, causally closed (*i.e.* left-closed for $\rightarrow^*$), and such that $\forall e \in E,\ e \in \kappa \Rightarrow e^\bullet \subseteq \kappa$. By convention we assume configurations also contain initial conditions C^0. See Fig. 4.6, right, for an example.

One immediately notices the one-to-one correspondence between configurations of an occurrence net and possible runs of that net, *i.e.* possible circulations of tokens. Specifically, a configuration must be read as a *partial order* of events. In Fig. 4.6 (right), the firing of t_1 precedes or is a cause to the firing of t_5, just like t_3, however, the ordering of t_1 and t_3 is not specified. So they can fire in any order. Every linear extension of the partial order defined by a configuration yields a valid firing sequence on its events. In that sense, a configuration represents an equivalence class of sequences, or a *trace* [27]. This is precisely where the true concurrency semantics allows to save with respect to the previous chapter: useless interleavings of concurrent events are not represented, and a single partial order replaces numerous sequences.

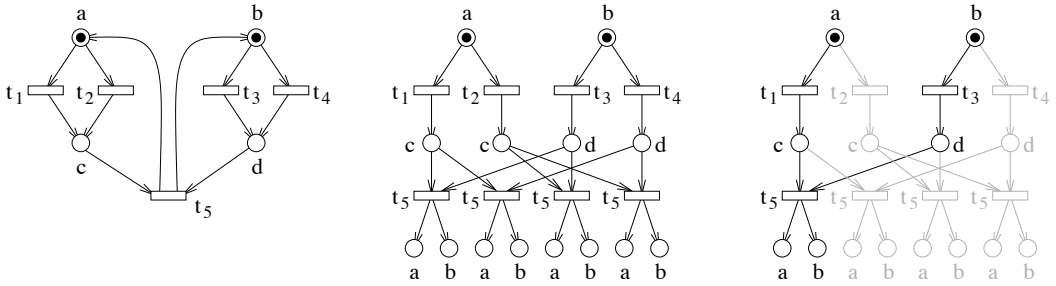


Figure 4.6: *An LMC net $\mathcal{N}$ (left), a branching process $\mathcal{O}$ of that net (center) and a configuration κ in $\mathcal{O}$ (right), corresponding to a run of $\mathcal{N}$. The folding of $\mathcal{O}$ into $\mathcal{N}$ is represented by transition and place names attached to events and conditions.*

LMC occurrence nets, equipped with morphisms of LMC nets, form the subcategory *Occ* of *Nets*. Observe that in this category, morphisms can only erase (not create) causality or conflict relations between two nodes. Concurrency relations are preserved, and configurations are mapped to configurations.

Branching process. The pair $(\mathcal{O}, f)$, where $f : \mathcal{O} \to \mathcal{N}$ is a morphism, is said to be a (labeled multi-clock) *branching process* (LMCBP or BP for short) of net $\mathcal{N}$ iff [31]

1. f is a total function on $\mathcal{O}$, also named a *folding* of $\mathcal{O}$,
2. $\forall e, e' \in E,\ [{}^\bullet e = {}^\bullet e',\ f(e) = f(e')] \ \Rightarrow\ e = e'$.

In other words, f labels all events and conditions of $\mathcal{O}$ by transitions and places of the net $\mathcal{N}$ (see Fig. 4.6). f being a morphism, this labeling actually turns a configuration κ of $\mathcal{O}$ into a run of the net $\mathcal{N}$. The parsimony condition 2. ensures that any run of $\mathcal{N}$ that is represented in $\mathcal{O}$ appears only once, or is represented by a unique configuration.

There exist similarities between the branching processes of LMC nets defined above for nets and the *sequential* branching processes defined for LMC automata. In particular the categorical architectures are identical, so the main results can be transported here. We briefly review them.

First of all, there exists a simple and intuitive algorithm to build a branching process $(\mathcal{O}, f)$ of net $\mathcal{N}$, that looks very much like Procedure 1 (without the merge of conditions). We briefly mention it below[4], since it forms the basis of several operations we use in the sequel.

Procedure 2

- Initialization :

[4]This recursive construction is explicit in [110] as well as in [35], where new events and places are named by a backward pointer technique. It also appears in the definition of "canonical branching processes" in [31].

- Create $|P^0|$ conditions in C^0, and define a bijection $f : C^0 \to P^0$.
- Set $C = C^0$, $E = \emptyset$ and $\rightarrow = \emptyset$.

• Recursion :

- Let X be a co-set of C and $t \in T$ a transition of $\mathcal{N}$ such that $f(X) = {}^\bullet t$.
- If there doesn't exist an event e in E with ${}^\bullet e = X$ and $f(e) = t$,
  * create a new event e in E with ${}^\bullet e = X$ and $f(e) = t$,
  * then create a subset Y of $|t^\bullet|$ new conditions in C, set $Y = e^\bullet$, and extend f to have the bijection $f : Y \to t^\bullet$.

The partition ν of conditions in $\mathcal{O}$ is of course inherited from the partition of places in $\mathcal{N}$, as well as the labeling λ of events.

Lemmas 3, 4 and 5 remain valid for branching processes of a net $\mathcal{N}$, and one easily modifies the recursion of Procedure 2 to compute the union of branching processes of $\mathcal{A}$, which leads to the notion of unfolding.

Unfolding. A *prefix* $\mathcal{O}'$ of an occurrence net $\mathcal{O}$ is defined as a sub-net of $\mathcal{O}$ which is causally closed, contains the initial marking (or equivalently all minimal conditions), and such that $\forall e \in E$, $e \in \mathcal{O}'$ implies $e^\bullet \subseteq \mathcal{O}'$. So a configuration of $\mathcal{O}$ is a conflict-free prefix of $\mathcal{O}$. The prefix relation is denoted by $\mathcal{O}' \sqsubseteq \mathcal{O}$.

Theorem 10 *Given a net $\mathcal{N}$, there exists a unique maximal branching process of $\mathcal{N}$ for the prefix relation. It is called the* unfolding *of $\mathcal{N}$. We denote it by $\mathcal{U}(\mathcal{N})$, or $\mathcal{U}_\mathcal{N}$ for short, and its corresponding folding by $f_\mathcal{N} : \mathcal{U}_\mathcal{N} \to \mathcal{N}$.*

A proof of this theorem can be found in Winskel's works [110] or in Engelfriet's [31].

The unfolding is of course obtained as the unique stationary point of Procedure 2, and its configurations yield all possible runs of $\mathcal{N}$.

Expressive power of branching processes. We had mentioned in the previous chapter that not all sub-languages $\mathcal{L}$ of an LMCA $\mathcal{A}$ could be represented as a sequential trellis (or branching) process. In the same way :

Lemma 8 *Let $\mathcal{L}$ be a set of configurations of $\mathcal{N}$. $\mathcal{L}$ can be represented as a BP $(\mathcal{O}, f)$ of $\mathcal{N}$ iff*

$$\forall \kappa \in \mathcal{L}, \qquad \kappa' \sqsubseteq \kappa \;\Rightarrow\; \kappa' \in \mathcal{L}$$
$$\forall \kappa_1, \kappa_2 \in \mathcal{L}, \qquad \kappa_1 \cup \kappa_2 \text{ is a configuration} \;\Rightarrow\; \kappa_1 \cup \kappa_2 \in \mathcal{L}$$

So ordinary BP represent languages that are both prefix-closed and closed by concurrent suffix extension. In the previous chapter (section 3.5.2), we introduced the notion of stop point for sequential branching process of $\mathcal{A}$, in order to get a one-to-one correspondence between sub-languages of $\mathcal{A}$ and runs encoded in a (stopped) branching process of $\mathcal{A}$. Similarly, to get a one-to-one correspondence between configuration sets of $\mathcal{N}$ and branching processes of $\mathcal{N}$, one must enrich the definition of BP with a notion of stop point. This takes the form of a function associating a

zero/one value to *cuts* of $\mathcal{O}$, *i.e.* configuration extremities. We don't detail it here: this mechanism is very heavy and almost kills the advantages of using branching processes.

4.3.2 Factorization property

Theorem 11 (universal property) *The unfolding $(\mathcal{U}_\mathcal{N}, f_\mathcal{N})$ of net $\mathcal{N}$ satisfies*

$$\forall \mathcal{O} \in Occ\,,\ \forall \phi : \mathcal{O} \to \mathcal{N},\ \exists! \psi : \mathcal{O} \to \mathcal{U}_\mathcal{N},\ \phi = f_\mathcal{N} \circ \psi \tag{4.9}$$

This property actually characterizes $\mathcal{U}_\mathcal{N}$ among all occurrence nets $\mathcal{O}$ that admit a morphism to $\mathcal{N}$ [110].

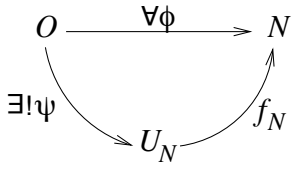


Figure 4.7: *Universal property of the unfolding of $\mathcal{N}$.*

Following the same category theory arguments as in the previous chapter (section 3.5.4), this universal property establishes a co-reflection between categories *Occ* and *Nets*. Specifically, the unfolding operation defines a functor $G = \mathcal{U} : Nets \to$

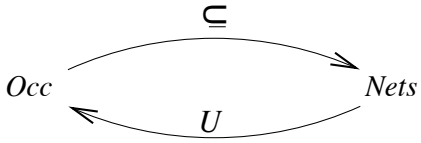


Figure 4.8: *Two functors relating Nets and Occ, that form an adjunction.*

Occ, and there exists an inclusion functor $F = \subseteq : Occ \to Nets$. They both form the adjoint pair (F, G), and since G has a left adjoint, we know it preserves categorical limits, in particular products and pullbacks. This entails once again:

$$\forall \mathcal{N}_1, \mathcal{N}_2 \in Nets, \qquad \mathcal{U}(\mathcal{N}_1 \times \mathcal{N}_2) \;=\; \mathcal{U}(\mathcal{N}_1) \times_O \mathcal{U}(\mathcal{N}_2) \tag{4.10}$$

where $\times_O$ denotes the categorical product in *Occ*. The existence of the latter is automatically granted, and it satisfies (or can be defined as)

$$\forall \mathcal{O}_1, \mathcal{O}_2 \in Occ, \qquad \mathcal{O}_1 \times_O \mathcal{O}_2 \;\simeq\; \mathcal{U}(\mathcal{O}_1 \times \mathcal{O}_2) \tag{4.11}$$

(4.11) is important in practice: it expresses that coupling Procedure 2 with the definition of the product in *Nets*, one can actually compute recursively the product of two occurrence nets. In the same manner, considering pullbacks, one has

$$\forall \mathcal{N}_0, \mathcal{N}_1, \mathcal{N}_2 \in Nets, \qquad \mathcal{U}(\mathcal{N}_1 \overset{\mathcal{N}_0}{\wedge} \mathcal{N}_2) \;=\; \mathcal{U}(\mathcal{N}_1) \overset{\mathcal{U}(\mathcal{N}_0)}{\wedge_O} \mathcal{U}(\mathcal{N}_2) \tag{4.12}$$

where the pullback $\wedge_O$ in Occ is defined by

$$\forall \mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2 \in Occ, \qquad \mathcal{O}_1 \overset{\mathcal{O}_0}{\wedge_O} \mathcal{O}_2 \;\simeq\; \mathcal{U}(\mathcal{O}_1 \overset{\mathcal{O}_0}{\wedge} \mathcal{O}_2) \tag{4.13}$$

and so $\wedge_O$ can again be computed recursively.

As we have seen in chapter 3, the factorization property of a structure encoding runs of a compound system is the first ingredient towards distributed diagnosis algorithms. So we are already in very good shape. The next ingredient is an adequate notion of projection, that we examine in the remainder of this chapter.

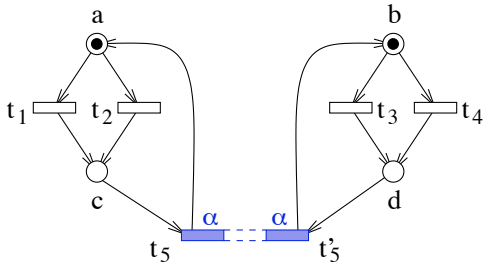


Figure 4.9: *The net in Fig. 4.6, expressed as a product of two components.*

But before, let us illustrate the interest of a relation like (4.10). The net $\mathcal{N}$ in Fig. 4.6, left, can be decomposed as the product of two smaller nets $\mathcal{N} = \mathcal{N}_1 \times \mathcal{N}_2$, see Fig. 4.9. Eq. (4.10) expresses that $\mathcal{U}_\mathcal{N}$ is thus the product, in the sense of occurrence nets, of the $\mathcal{U}_{\mathcal{N}_i}$ depicted in Fig. 4.10. The latter are simple trees, since the $\mathcal{N}_i$ are automata. One first notices that unfoldings grow in width as one progresses in "length" (the time dimension). This is precisely one of the drawbacks addressed by the next chapter. Now, considering the number of possibilities to fire t_5 (resp. t'_5) in $\mathcal{U}_{\mathcal{N}_1}$ (resp. $\mathcal{U}_{\mathcal{N}_2}$), one finds two solutions for the first occurrence of t_5, then four for the second occurrence, etc. In the expanded product $\mathcal{U}_\mathcal{N}$, they result in $2 \times 2 = 4$ possibilities to fire the first occurrence of (t_5, t'_5) (see Fig. 4.6, right), then $16 = 4 \times 4$ for the second occurrence, etc. In other words, the factorized form of $\mathcal{U}_\mathcal{N}$ is more compact than the expanded one, just like $(a + b + c + ...)^p$ is more compact than its expanded form. Therefore it is likely that processings based on the factorized form of $\mathcal{U}_\mathcal{N}$ will be more efficient than processings on the expanded form, as it was already the case in chapter 3 with sequential semantics.

4.4 Distributed diagnosis : the unfolding approach

As soon as the product is defined in Occ, a natural notion of projection becomes available. We first explore how far one can go with this notion, in terms of modular computations for the diagnosis problem.

Notations : notice that we denote by $\mathcal{O}^b$ the observations, and by $\mathcal{O}$ general occurrence nets or branching processes.

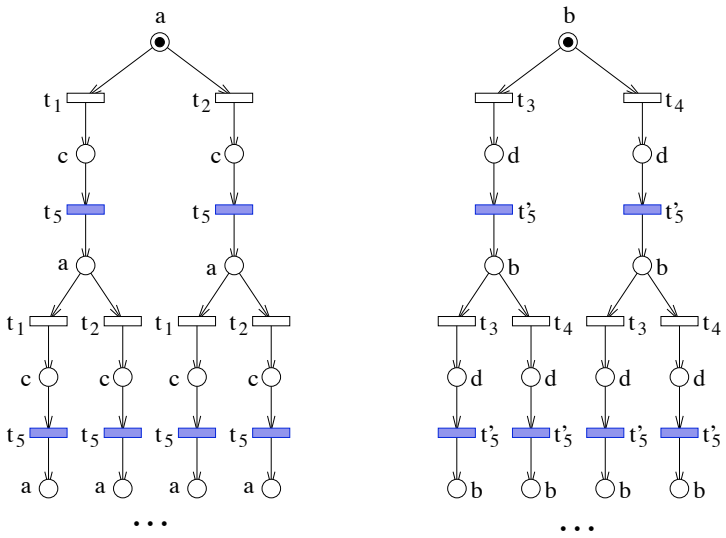


Figure 4.10: *Unfoldings of the two components of Fig. 4.9.*

4.4.1 Projection

Natural projection. Let us first consider two nets $\mathcal{N}_1, \mathcal{N}_2$, and branching processes $(\mathcal{O}_i, f_i)$ of these nets. The occurrence net $\mathcal{O} = \mathcal{O}_1 \times_O \mathcal{O}_2$ is associated to two morphisms $\psi_i : \mathcal{O} \to \mathcal{O}_i$. Combined to the foldings f_i, they yield the $f_i \circ \psi_i : \mathcal{O} \to \mathcal{N}_i$, and since one has morphisms from $\mathcal{O}$ to the $\mathcal{N}_i$, by the universal property of $\mathcal{N} = \mathcal{N}_1 \times \mathcal{N}_2$ (prop. 9), there exists a unique $f : \mathcal{O} \to \mathcal{N}$ such that $f_i \circ \psi_i = f \circ \phi_i$, where $\phi_i : \mathcal{N} \to \mathcal{N}_i$. It is easily checked that f is a folding, which makes $\mathcal{O}_1 \times_O \mathcal{O}_2$ a branching process of $\mathcal{N}_1 \times \mathcal{N}_2$ (Fig. 4.11).

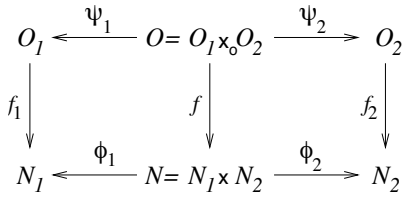


Figure 4.11: *The product of two branching processes yields a branching process of the product.*

We take as projections of $\mathcal{O}$ the two occurrence nets $\mathcal{O}'_i = \psi_i(\mathcal{O})$. They are prefixes of the original factors $\mathcal{O}_i$ ($\mathcal{O}'_i \sqsubseteq \mathcal{O}_i$) and satisfy $\mathcal{O} = \mathcal{O}'_1 \times_O \mathcal{O}'_2$. Referring to notions introduced in the previous chapter (end of section 3.1.2), the $\mathcal{O}'_i$ thus form the *minimal product covering* of $\mathcal{O}$, in the sense that taking a strictly smaller prefix $\mathcal{O}''_i \sqsubset \mathcal{O}'_i$ of one factor kills the equality.

This is illustrated by the example in Fig. 4.12, that depicts the product of two configurations (κ_1, left and κ_2, right). The product lies in the middle, and all labels $\{\alpha, \beta\}$ are supposed shared. Observe that the event t_3 of κ_1, labeled by β, finds no partner in κ_2 and thus vanishes. So $\psi_1(\kappa_1 \times_O \kappa_2)$ reduces to events $\mathbf{t}_1$ and $\mathbf{t}_2$. This example also illustrates that the product of two configuration is not necessarily a

configuration : there may exist different associations of their events. Finally, notice that the two concurrent events t_1, t_2 of κ_1 inherit a causality relation of κ_2 in the product, either in one direction or the other.

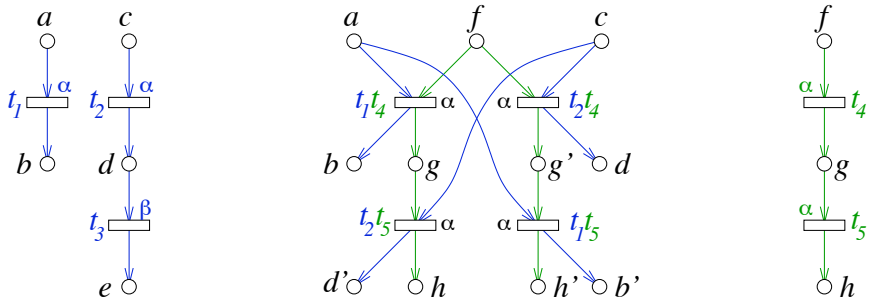


Figure 4.12: *Product (center) of two configurations (left and right), that share all event labels,* i.e. α *and* β*.*

To define the projection of any branching process of $\mathcal{N} = \mathcal{N}_1 \times \mathcal{N}_2$, we consider the same scheme with $\mathcal{U}_\mathcal{N}$ instead of $\mathcal{O}$. From $\mathcal{U}_\mathcal{N} = \mathcal{U}_{\mathcal{N}_1} \times_O \mathcal{U}_{\mathcal{N}_2}$ stated in (4.10), and its associated morphisms $\psi_i : \mathcal{U}_\mathcal{N} \to \mathcal{U}_{\mathcal{N}_i}$, one can define the projections $\mathcal{O}'_i = \psi_i(\mathcal{O})$ for any branching process $\mathcal{O} \subseteq \mathcal{U}_\mathcal{N}$ of $\mathcal{N}$. One has $\mathcal{O}'_i \subseteq \mathcal{U}_{\mathcal{N}_i}$ which obviously makes them branching processes of the $\mathcal{N}_i$. They form the minimal product covering of $\mathcal{O}$ in the sense that $\mathcal{O} \sqsubseteq \mathcal{O}'_1 \times_O \mathcal{O}'_2$, and replacing one of the factors $\mathcal{O}'_i$ by a strict prefix $\mathcal{O}''_i \sqsubset \mathcal{O}'_i$ kills this prefix inclusion of $\mathcal{O}$. So the projection keeps events of the $\mathcal{U}_{\mathcal{N}_i}$ that are strictly useful to $\mathcal{O}$, and only them. Observe that $\mathcal{O}$ is included in its product covering (whence the name), and is equal to it iff $\mathcal{O}$ was already in product form.

Projection operator. We now move to the general case $\mathcal{N} = \mathcal{N}_1 \times \ldots \times \mathcal{N}_N$ and consider $\mathcal{O} = \mathcal{O}_1 \times_O \ldots \times_O \mathcal{O}_N$ where each $\mathcal{O}_i$ is a BP of $\mathcal{N}_i$. For $I \subseteq \{1, \ldots, N\}$, we still use notations $\mathcal{N}_I = \times_{i \in I} \mathcal{N}_i$ and $\mathcal{O}_I = \times^O_{i \in I} \mathcal{O}_i$. By associativity of $\times_O$, we can write $\mathcal{O} = \mathcal{O}_I \times_O \mathcal{O}_{I^c}$ and define $\Pi_I(\mathcal{O})$ as $\psi_I(\mathcal{O})$. This projection operation generalizes to any $\mathcal{O}_J$, for $J \subseteq \{1, \ldots, N\}$, by $\Pi_I(\mathcal{O}_J) \triangleq \Pi_{I \cap J}(\mathcal{O}_J)$. And by associativity of $\times_O$, one easily checks that

$$\forall I, J \subseteq \{1, \ldots, N\}, \quad \Pi_I \circ \Pi_J \quad = \quad \Pi_{I \cap J} \tag{4.14}$$

which corresponds to axiom (a1).

In practice, it is quite easy to compute the projection of a BP : to get $\Pi_I(\mathcal{O}_J)$, it suffices to remove conditions and events not labeled by $\mathcal{N}_I$, and to "trim" the result, *i.e.* to recursively merge isomorphic events in order to satisfy the parsimony criterion (requirement 2 in the definition of a BP).

Minimal product covering. This notion applies as above :

$$\forall \mathcal{O} \sqsubseteq \mathcal{U}_\mathcal{N}, \qquad \mathcal{O} \sqsubseteq \mathcal{O}'_1 \times_O \ldots \times_O \mathcal{O}'_N \quad \text{where} \quad \mathcal{O}'_i = \Pi_i(\mathcal{O}) \sqsubseteq \mathcal{U}_{\mathcal{N}_i} \tag{4.15}$$

$$\forall \mathcal{O}''_i \sqsubseteq \mathcal{U}_{\mathcal{N}_i}, \ \exists j : \mathcal{O}''_j \sqsubset \mathcal{O}'_j \quad \Rightarrow \quad \mathcal{O} \not\sqsubseteq \mathcal{O}''_1 \times_O \ldots \times_O \mathcal{O}''_N \tag{4.16}$$

In (4.15), one has equality iff $\mathcal{O}$ admits a product form: $\mathcal{O} = \mathcal{O}_1 \times_O \ldots \times_O \mathcal{O}_N$, and in that case $\mathcal{O}'_i \sqsubseteq \mathcal{O}_i$. Moreover, one has $\mathcal{O}'_I = \Pi_I(\mathcal{O}) = \times^O_{i\in I} \mathcal{O}'_i$ with $\mathcal{O}'_i = \Pi_i(\mathcal{O})$.

4.4.2 Centralized diagnosis

Single sensor. The setting is like in the previous chapter: net $\mathcal{N}$ produces a hidden run κ, that is only observed through the labels of $\Lambda' \subseteq \Lambda$ that it produces. But what does it mean to observe the partial order of labels $\lambda(\kappa)_{|\Lambda'}$?

Things become clear when one assumes that labels are logged in sequence by a sensor. So instead of $\lambda(\kappa)_{|\Lambda'}$ one actually sees a linear extension of this partial order, under the assumption that the observation process is not anti-causal: visible events that are causally related can't lead to labels observed in the reversed order. However, two concurrent events could produce labels observed in any order.

Now, the observed labels may not all be ordered: indeed, when two visible labels are collected on different sensors, the possible causal link relating these events is generally lost, unless some specific mechanism is deployed to preserve it. Therefore, as a general situation we assume that our observation $\mathcal{O}^b$ is a weakened version of an extension of $\lambda(\kappa)_{|\Lambda'}$ (see Fig. 4.13 for an example).

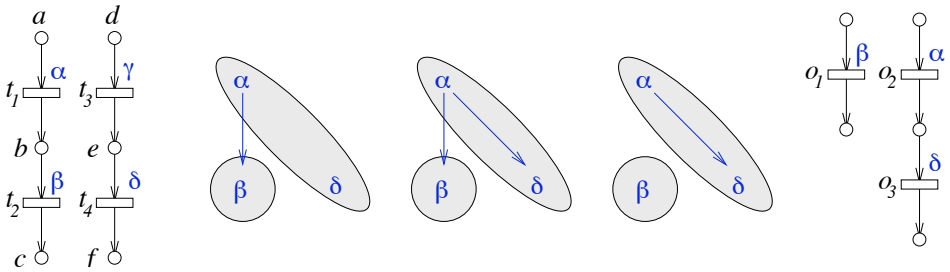


Figure 4.13: *A hidden configuration κ (left), and an observed partial order of its labels, expressed as a configuration (right). γ is supposed to be invisible,* i.e. $\Lambda' = \{\alpha, \beta, \delta\}$. *The second plot is the partial order on visible labels induced by κ. The ordering of α and δ (third plot) may be created by the sensor that collects them. By contrast, the causality relation $\alpha \rightarrow \beta$ may be lost if these labels are collected by different sensors (fourth plot).*

As a partial order of labels, $\mathcal{O}^b$ can be encoded under the form of a configuration (a LMCON), and a run κ of $\mathcal{N}$ is a valid explanation of $\mathcal{O}^b$ if it perfectly synchronizes with $\mathcal{O}^b$, *i.e.* if $\psi^b_{\mathcal{O}}(\kappa \times_O \mathcal{O}^b) = \mathcal{O}^b$. To compute them all at once, consider

$$\mathcal{D} \quad = \quad \mathcal{U}_{\mathcal{N}} \times_O \mathcal{O}^b \tag{4.17}$$

Not all maximal configurations of $\mathcal{D}$ explain entirely $\mathcal{O}^b$, they may explain only a prefix of $\mathcal{O}^b$ and then be blocked. But those that do explain all $\mathcal{O}^b$ form a configuration set closed by concurrent extension (see lemma 8). In other words, it suffices to cut off dead branches of $\mathcal{D}$, then take the remaining maximal configurations to get all solutions to the diagnosis problem, once projected back on $\mathcal{U}_{\mathcal{N}}$. For simplicity, we assume in the sequel that the diagnosis is given by (4.17) above.

Multiple sensors. We now assume that N sensors are collecting labels produced by transitions of net $\mathcal{N}$, and possibly observe the same labels. For example sensor i reacts to labels of Λ'_i and produces configuration $\mathcal{O}_i^b$ as observation. Specifically, $\mathcal{O}_i^b$ is a weakened version of some extension of $\lambda(\kappa)$, restricted to Λ'_i. It is important to notice that the same extension of $\lambda(\kappa)$ lies behind all the $\mathcal{O}_i^b$, otherwise the sensors could observe in different orders two concurrent events, which would lead to an inconsistency. We easily get back to the previous case by defining

$$\mathcal{O}^b \quad = \quad \mathcal{O}_1^b \times_O \ldots \times_O \mathcal{O}_N^b \tag{4.18}$$

as our joint observation.

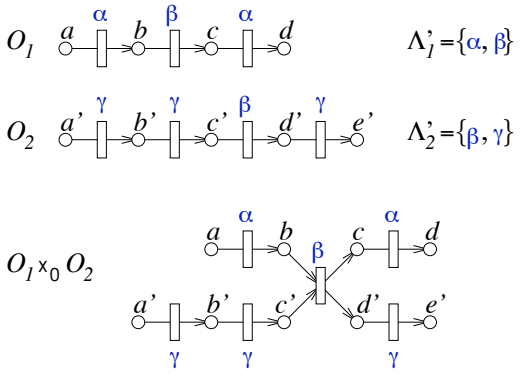


Figure 4.14: *Two observed configurations, on sensors sharing label β, and the resulting equivalent joint observation.*

This deserves several comments. First, observe that the $\mathcal{O}_i^b$ synchronize on the labels they jointly observe. If the Λ'_i are disjoint, putting all observations together simply amounts to juxtaposing the $\mathcal{O}_i^b$, which is much simpler than the complex computation of all interleavings of Fig. 3.18. Translated in the true concurrency semantics, the latter becomes Fig. 4.14. Secondly, taking as observation the product of the $\mathcal{O}_i^b$ may yield a general occurrence net, not necessarily a configuration (see Fig. 4.12). This doesn't change much the theory: it suffices to take as solution to the diagnosis problem a maximal configuration of $\mathcal{D}$ that projects into a maximal configuration of $\mathcal{O}^b$, equivalently into maximal configurations of the $\mathcal{O}_i^b$ (*i.e.* the $\mathcal{O}_i^b$ themselves).

4.4.3 Distributed diagnosis

Algebraically, the situation is pretty much the same as for sequential semantics (section 3.6.3). We now assume that the supervised system $\mathcal{N}$ is obtained by assembling smaller nets $\mathcal{N}_i$, that we call *sites*: $\mathcal{N} = \mathcal{N}_1 \times \ldots \times \mathcal{N}_N$. Each of the $\mathcal{N}_i$ is attached to a specific sensor collecting labels of $\Lambda'_i \subseteq \Lambda_i$ and producing observation $\mathcal{O}_i^b$, as above. By injecting (4.10) and (4.18) into (4.17), one gets

$$\mathcal{D} \quad = \quad [\mathcal{U}_{\mathcal{N}_1} \times_O \mathcal{O}_1^b] \times_O \ldots \times_O [\mathcal{U}_{\mathcal{N}_N} \times_O \mathcal{O}_N^b] \tag{4.19}$$

This relation expresses that the global diagnosis corresponds to the synchronization of local diagnoses $\mathcal{D}_i = \mathcal{U}_{\mathcal{N}_i} \times_O \mathcal{O}_i^b$ computed for each site. Our objective is to derive the minimal product covering of $\mathcal{D}$, that is the branching processes $\mathcal{O}_i = \Pi_{\mathcal{N}_i}(\mathcal{D})$, or equivalently the $\mathcal{D}'_i = \Pi_{\mathcal{N}_i \times \mathcal{O}_i^b}(\mathcal{D}) = \mathcal{O}_i \times_O \mathcal{O}_i^b$. They correspond to coherent local views of $\mathcal{D}$, in the sense that they satisfy

$$\begin{aligned} \mathcal{D} &= [\mathcal{O}_1 \times_O \mathcal{O}_1^b] \times_O \ldots \times_O [\mathcal{O}_N \times_O \mathcal{O}_N^b] \\ &= \mathcal{D}'_1 \times_O \ldots \times_O \mathcal{D}'_N \end{aligned} \tag{4.20}$$

Of course, a crucial point is to derive the $\mathcal{O}_i$ or the $\mathcal{D}'_i$ without computing $\mathcal{D}$ itself, since the latter can be a huge object (recall that products generally multiply transitions). So we aim at deploying the modular computation formalism of chapter 2.

Translation into pullbacks. As in the case of sequential semantics (section 3.6.3), we don't yet have a clean theory that would allow to project a branching process on a label set, and would allow as well to combine the resulting object (probably an event structures carrying labels) with another branching process[5]. By contrast, there exists a natural notion of projection on sites, which suggests to use pullbacks to transmit information from component to component, see (3.44).

We thus restate $\mathcal{N}$ as $\mathcal{N} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_M$ where *components* $\mathcal{S}_j$ are defined by $\mathcal{S}_j = \mathcal{N}_{I_j} = \times_{i \in I_j} \mathcal{N}_i$ and cover all the sites: $\cup_j I_j = \{1, \ldots, N\}$. One has

$$\mathcal{U}_{\mathcal{N}} = \mathcal{U}_{\mathcal{S}_1} \wedge_O \ldots \wedge_O \mathcal{U}_{\mathcal{S}_M} \tag{4.21}$$

and similarly

$$\begin{aligned} \mathcal{D} &= [\mathcal{U}_{\mathcal{S}_1} \times_O \mathcal{O}_{I_1}^b] \wedge_O \ldots \wedge_O [\mathcal{U}_{\mathcal{S}_M} \times_O \mathcal{O}_{I_M}^b] \\ &= \mathcal{D}_1 \wedge_O \ldots \wedge_O \mathcal{D}_M \end{aligned} \tag{4.22}$$

where each local diagnosis $\mathcal{D}_j$ incorporates by $\mathcal{O}_{I_j}^b$ observations on all sites $\mathcal{N}_i$ of component $\mathcal{S}_j$. It is not necessary to duplicate observations $\mathcal{O}_i^b$ when site $\mathcal{N}_i$ appears in several components. It suffices to "distribute" the $\mathcal{O}_i^b$ on components, in such a way that each $\mathcal{O}_i^b$ appears at least once in (4.22). But taking all observations $\mathcal{O}_{I_j}^b$ on $\mathcal{S}_j$ has the advantage of " maximally reducing" $\mathcal{U}_{\mathcal{S}_j}$, an interesting property for modular computations.

The distributed diagnosis problem now amounts to computing the $\mathcal{O}_{I_j} = \Pi_{I_j}(\mathcal{D})$, or equivalently the $\mathcal{D}'_j = \Pi_{S_j \times_O \mathcal{O}_{I_j}^b}(\mathcal{D}) = \mathcal{O}_{I_j} \times_O \mathcal{O}_{I_j}^b$, that form the minimal pullback covering of $\mathcal{D}$:

$$\begin{aligned} \mathcal{D} &= [\mathcal{O}_{I_1} \times_O \mathcal{O}_{I_1}^b] \wedge_O \ldots \wedge_O [\mathcal{O}_{I_M} \times_O \mathcal{O}_{I_M}^b] \\ &= \mathcal{D}'_1 \wedge_O \ldots \wedge_O \mathcal{D}'_M \end{aligned} \tag{4.23}$$

If necessary, the $\Pi_{\mathcal{N}_i}(\mathcal{D})$ of the minimal product covering can be deduced from the larger $\mathcal{D}'_j$.

[5]This can probably be done, but remains a research topic.

Misleading projections. This notion underlines a problem that arises with natural projections defined on branching processes. Computing projection $\mathcal{O}'_I = \Pi_I(\mathcal{O})$ amounts, in part, to selecting some events and conditions in $\mathcal{O}$. But this operation may forget causalities or conflicts that were linking them (see. Fig. 4.15). As a result, two events or conditions may unduly appear as concurrent after projection, and thus be allowed to appear in the same configuration, or with a reversed causality, whereas this would have been impossible in the original branching process $\mathcal{O}$.

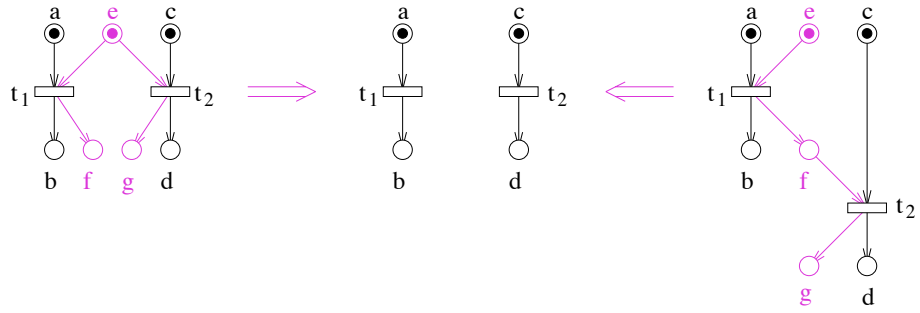


Figure 4.15: *Two branching processes (left and right) that create a fake concurrency (center) once the influence of the central variable is discarded by the projection. The BP on the left loses a conflict relation, the one on the right loses a causality relation.*

We say that $\mathcal{O}'_I = \Pi_I(\mathcal{O})$ is *not misleading* iff every configuration κ' in $\mathcal{O}'_I$ can be obtained as the projection of a configuration κ of $\mathcal{O}$. One has:

Lemma 9 *Let us say that $\mathcal{N}_i$ is a simple site if, as a LMC net, it is composed of a single state variable. Projections of branching processes on simple sites are never misleading.*

This is due to the fact that a BP of a simple site can't have concurrent events. The next section is precisely devoted to defining a notion of projection that keeps track of useful conflicts and causalities.

Separation theorem. We are now well equipped to state the separation theorem, that forms the backbone of modular computations.

Theorem 12 *Let $I, K \subseteq \{1, \ldots, N\}$, and take $J = I \cap K$. Let $\mathcal{O}_I, \mathcal{O}_K$ be branching processes of $\mathcal{N}_I, \mathcal{N}_K$ respectively, and assume that shared sites $\mathcal{N}_J$ capture all interaction labels between $\mathcal{N}_I$ and $\mathcal{N}_K$,* i.e. *$\Lambda_I \cap \Lambda_K \subseteq \Lambda_J$. If projections Π_J are not misleading, then*

$$\Pi_J(\mathcal{O}_I \wedge_O \mathcal{O}_K) \quad = \quad \Pi_J(\mathcal{O}_I) \wedge_O \Pi_J(\mathcal{O}_K) \tag{4.24}$$

This weak form of axiom (a3) reproduces the formal setting of section 3.6.3, so we can still derive propagation and merge primitives:

Proposition 11 *Let $I, J, K \subseteq \{1, \ldots, N\}$, and let $\mathcal{O}_I, \mathcal{O}_J, \mathcal{O}_K$ be branching processes of $\mathcal{N}_I, \mathcal{N}_J, \mathcal{N}_K$ respectively. If $\Lambda_I \cap \Lambda_K \subseteq \Lambda_J$, denoted $I|J|K$, and if projections Π_J are not misleading, then*

$$\Pi_J(\mathcal{O}_I \wedge_O \mathcal{O}_J \wedge_O \mathcal{O}_K) = \Pi_J(\mathcal{O}_I \wedge_O \mathcal{O}_J) \wedge_O \Pi_J(\mathcal{O}_J \wedge_O \mathcal{O}_K) \quad (4.25)$$

$$\Pi_I(\mathcal{O}_I \wedge_O \mathcal{O}_J \wedge_O \mathcal{O}_K) = \Pi_I[\mathcal{O}_I \wedge_O \Pi_J(\mathcal{O}_J \wedge_O \mathcal{O}_K)] \quad (4.26)$$

Compared to the chapter on sequential semantics, the essential change is thus the new ingredient of "non misleading" projections. To show the importance of this assumption, let us examine the counter-example of Fig. 4.16. The two BPs $\mathcal{O}_1$ (left) and $\mathcal{O}_2$ (right) have the same projection on variables $\bar{a}$ and $\bar{c}$ (center), which allows to fire t_1 and t_2 concurrently. However, in reality $\mathcal{O}_1$ imposes to fire t_1 *before* t_2, while $\mathcal{O}_2$ imposes the converse, so their pullback $\mathcal{O}_1 \wedge_O \mathcal{O}_2$ contains no event at all and reduces to the initial conditions $\{a, e, c, h\}$.

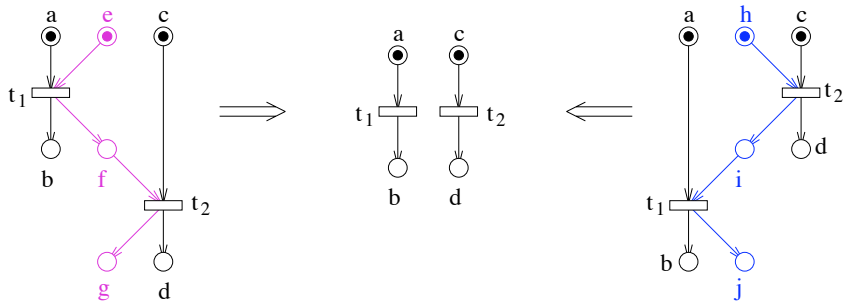


Figure 4.16: *With misleading projections, theorem 12 doesn't hold.*

Support graph for modular computations. Identical to section 3.6.3.

4.4.4 Example

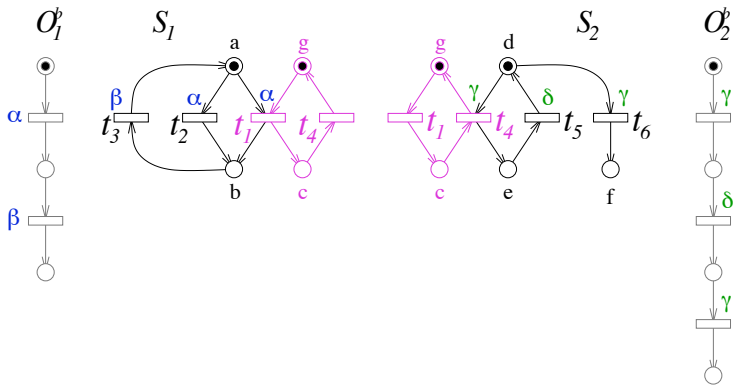


Figure 4.17: *Two components $\mathcal{S}_1 = \mathcal{N}_1 \times \mathcal{N}_3$ and $\mathcal{S}_2 = \mathcal{N}_3 \times \mathcal{N}_2$ formed of two sites each, and observations $\mathcal{O}_1^b, \mathcal{O}_2^b$ on these components.*

The center of Fig. 4.17 represents two components $\mathcal{S}_1 = \mathcal{N}_1 \times \mathcal{N}_3, \mathcal{S}_2 = \mathcal{N}_3 \times \mathcal{N}_2$ that share site $\mathcal{N}_3$, in charge of flipping tokens from g to c. Their pullback amounts to superimposing this common sub-net. With a slight abuse of notations, we assume the pullback is performed taking transition names as labels. Transitions of these nets produce labels (Greek letters) that have been collected under the form of the two sequences $\mathcal{O}_1^b$ and $\mathcal{O}_2^b$. On this example, we assume that the labels used for the pullback differ from those collected by sensors, so nets are provided with two label sets (or equivalently with a synchronization algebra [111]), which doesn't change much the theory but introduces some flexibility.

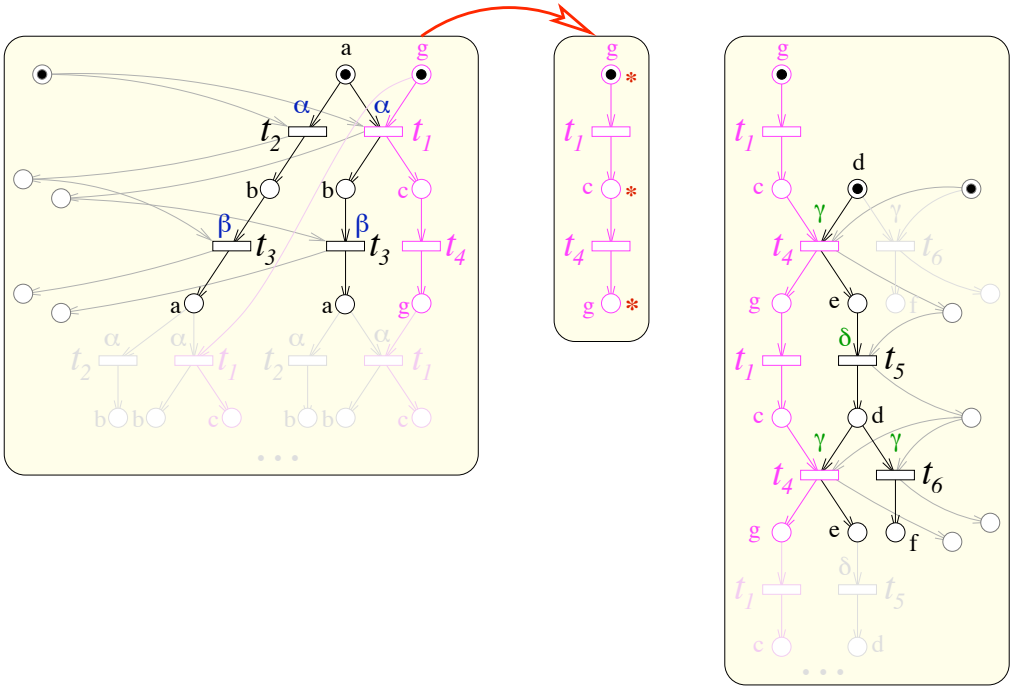


Figure 4.18: *Local diagnoses* $\mathcal{D}_i = \mathcal{U}_{\mathcal{S}_i} \times_O \mathcal{O}_i^b$ *(left and right), and the message* $\Pi_3(\mathcal{D}_1)$ *from* $\mathcal{S}_1$ *to* $\mathcal{S}_2$ *(center).*

The local diagnoses $\mathcal{D}_1, \mathcal{D}_2$ are depicted in Fig. 4.18. They are obtained by unfolding each $\mathcal{S}_i$ and taking the product with the corresponding $\mathcal{O}_i^b$. Events and conditions in gray are discarded by the product. Observe that in $\mathcal{D}_2$ the first firing of t_6 is discarded although it matches the first observation γ. The reason is that t_6 has no future, and so can't lead to an explanation for the next observation δ. This corresponds to a dead branch in $\mathcal{U}_{\mathcal{S}_2} \times_O \mathcal{O}_2$. The BP in the middle represents the projection of $\mathcal{D}_1$ on the interface net $\mathcal{N}_3$, which is the *message* from $\mathcal{S}_1$ to $\mathcal{S}_2$. It essentially expresses that observations $\mathcal{O}_1^b$ do not allow a second firing of $\mathbf{t}_1$. The stars next to conditions indicate possible stop points[6], *i.e.* projections of configurations that explain all observations $\mathcal{O}_1^b$. Once inserted into $\mathcal{D}_2$, by a pullback operation, this message removes some of the explanations: a single configuration remains in $\mathcal{D}_2'$ (Fig. 4.19).

[6] As previously mentioned, the notion of stop point for BP is a 0/1 function on cuts of this BP.

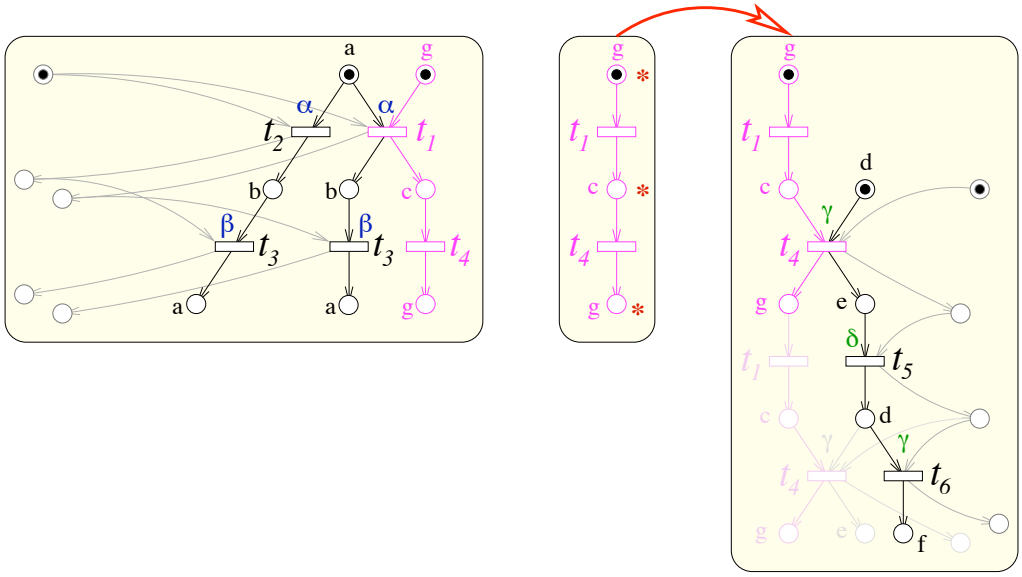


Figure 4.19: *Insertion of the message* $\Pi_3(\mathcal{D}_1)$ *into* $\mathcal{D}_2$ *by* $\mathcal{D}_2' = \Pi_3(\mathcal{D}_1) \wedge_O \mathcal{D}_2$.

The message in the reverse direction (Fig. 4.20) is longer and also differs by the position of stop points: $\mathcal{D}_2$ can't stop before the first t_4 is fired. Once inserted into $\mathcal{D}_1$, this message renders impossible the solution (t_2, t_3) because this private trajectory doesn't provide the sequence (t_1, t_2) to $\mathcal{S}_2$. So again a single configuration remains possible in $\mathcal{D}_1'$.

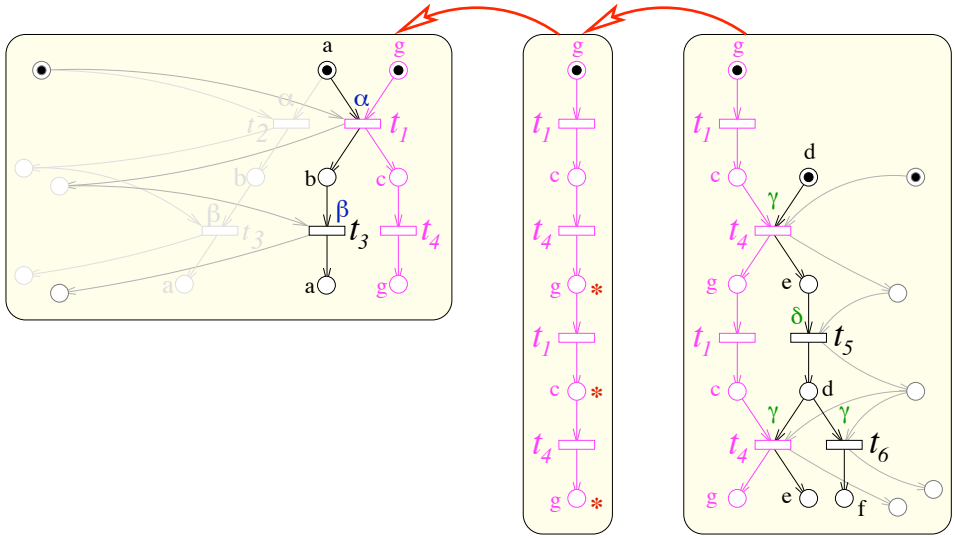


Figure 4.20: *Message from* $\mathcal{S}_2$ *to* $\mathcal{S}_1$*, and its insertion into* $\mathcal{D}_1$ *by* $\mathcal{D}_1' = \Pi_3(\mathcal{D}_2) \wedge_O \mathcal{D}_1$.

Notice that putting together the reduced local diagnoses yields a unique solution to the problem: $\mathcal{D} = \mathcal{D}_1' \wedge_O \mathcal{D}_2'$ contains a single maximal configuration (Fig. 4.21). The latter reveals that α necessarily appeared before the first γ in $\mathcal{O}_2^b$, while β is concurrent with $\mathcal{O}_2^b$. Moreover, the observed sequences do correspond to true

causalities in the underlying system. So we have rediscovered the causality relations of our observations.

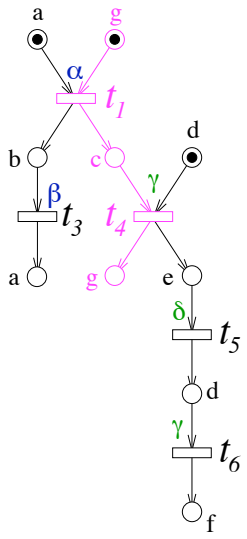


Figure 4.21: *Global solution to the diagnosis problem.*

Of course, this example doesn't manifest the interest of modular computations : the factorized form $\mathcal{U}_{\mathcal{S}_1} \wedge_O \mathcal{U}_{\mathcal{S}_2}$ has the same complexity as the expanded form $\mathcal{U}_{\mathcal{S}}$. The advantage appears when the two systems have different ways of producing and consuming the resources of their interface $\mathcal{N}_3$.

4.4.5 Involutivity

MC branching processes are involutive, *i.e.* satisfy $\mathcal{O} \wedge_O \Pi_I(\mathcal{O}) = \mathcal{O}$. So we are in the setting where message passing algorithms converge on any graph, and produce an approximation of the reduced local diagnoses. Instead of the $\mathcal{D}'_j = \Pi_{\mathcal{S}_j \times_O \mathcal{O}^b_{I_j}}(\mathcal{D})$, one gets the upper approximations $\mathcal{D}''_j$ that satisfy

$$\forall i, \mathcal{D}'_j \sqsubseteq \mathcal{D}''_j \sqsubseteq \mathcal{D}_j \quad \text{and} \quad \mathcal{D} = \mathcal{D}''_1 \wedge_O \ldots \wedge_O \mathcal{D}''_M \tag{4.27}$$

It can also be shown that MPA perform a progressive reduction (theorem 4), and that the limit $\mathcal{D}''_M$ satisfies the local extendibility property (theorem 3).

Misleading projections. What happens if the presence of misleading projections is ignored ? Theorem 12 is not completely lost : instead of (4.24), one gets

$$\Pi_J(\mathcal{O}_I \wedge_O \mathcal{O}_K) \quad \sqsubseteq \quad \Pi_J(\mathcal{O}_I) \wedge_O \Pi_J(\mathcal{O}_K) \tag{4.28}$$

which means that message passing algorithms do propagate constraints, but not all of them. This is due to the fact that misleading projections create fake concurrencies, and thus fake configurations. MPA still converge on trees in a finite number of (useful) steps. In reason of involutivity, convergence is also granted on any graph. And in both cases the limit $\mathcal{D}''_j$ satisfies (4.27).

4.5 Augmented branching processes

[The results described in this section are not yet published, at least under this form. Part of these ideas appeared in [41] under the form of event structure based computations.]

The BP computations presented in the previous section remain valid as long as projections are not misleading, *i.e.* do not create fake concurrency, and consequently fake configurations. This phenomenon appears when projections erase components/sites that were responsible for a conflict (Fig. 4.15 left) or a causal link (Fig. 4.15 right). In order to avoid the creation of fake runs after projection, one would like to preserve such conflicts or causal links between events, when they are necessary. Augmented branching processes are designed for this purpose: in addition to the standard elements of a BP, they also carry extra causal links and extra conflicts, as illustrated in Fig. 4.22.

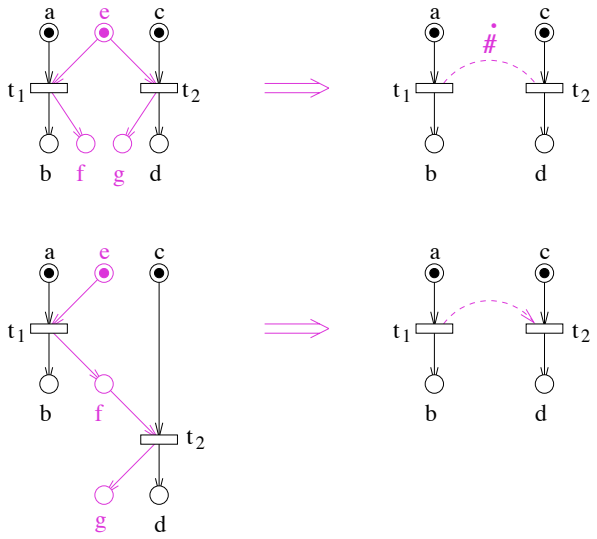


Figure 4.22: *Preservation of conflicts and causalities in projections.*

4.5.1 Definition

Augmented occurrence net. A (labeled multi-clock) *augmented occurrence net* (AON for short) $\dot{\mathcal{O}} = (C, E, \rightarrow, C^0, \nu, \lambda, \Lambda, \dot{\prec}, \dot{\#})$ is obtained by adjoining a causality and a conflict relation to the occurrence net $\mathcal{O} = (C, E, \rightarrow, C^0, \nu, \lambda, \Lambda)$, in such a way that

1. $\dot{\prec}$ is a well founded partial order relation on E extending the partial order $\prec \triangleq \rightarrow^+$ of $\mathcal{O}$: $\quad \forall\, e, e' \in E, \quad e \prec e' \;\Rightarrow\; e \,\dot{\prec}\, e'$

2. $\dot{\#}$ is a symmetric and anti-reflexive relation on E, extending the natural conflict $\#$ of $\mathcal{O}$: $\forall\, e, e' \in E, \;\; e \# e' \Rightarrow e \,\dot{\#}\, e'$,

3. $\dot{\#}$ is inherited *via* causality $\dot{\prec}$: $\;\forall\, e, e', e'' \in E, \quad e \,\dot{\#}\, e'$ and $e' \,\dot{\prec}\, e'' \;\Rightarrow\; e \,\dot{\#}\, e''$

The augmentation of $\mathcal{O}$ with $\dot{\prec}$ and $\dot{\#}$ amounts to replacing some concurrency links between events either by a causality or by a conflict. We call $\dot{\#} \setminus \#$ (resp. $\dot{\prec} \setminus \prec$) the extra conflict (resp. causality) relations. Notice that $(E, \dot{\prec}, \dot{\#})$ is a regular event structure, therefore the objects we have introduced are a mixture between event structures and occurrence nets. The importance of preserving conditions will appear in the definition of projections.

Relations $\dot{\prec}$ and $\dot{\#}$ are defined on E but naturally propagate to all nodes of $\dot{\mathcal{O}}$, by transitivity for $\dot{\prec}$ and by inheritance for $\dot{\#}$. As for standard occurrence nets, events e_1 and e_2 are said to be concurrent, denoted by $e_1 \dot{\perp\!\!\!\perp} e_2$, iff $\neg(e_1 \dot{\prec} e_2)$, $\neg(e_2 \dot{\prec} e_1)$ and $\neg(e_1 \dot{\#} e_2)$. Co-sets and cuts are defined from $\dot{\perp\!\!\!\perp}$ in the usual way.

Prefix. The notion of *prefix* differs slightly from the one defined for standard occurrence nets. The AON $\dot{\mathcal{O}}' = (C', E', \rightarrow, C^0, \nu, \lambda, \Lambda, \dot{\prec}, \dot{\#}')$ is a prefix of $\dot{\mathcal{O}}$, still denoted by $\dot{\mathcal{O}}' \sqsubseteq \dot{\mathcal{O}}$, iff

- C' and E' are left causally closed in $\dot{\mathcal{O}}$ for $\dot{\prec}$, and $\forall e \in E, [e \in E' \Rightarrow e^\bullet \in C']$,
- $\dot{\#}'$ *contains* the restriction of $\dot{\#}$ to E'.

It is therefore sufficient to *reinforce* the conflict relation $\dot{\#}$ of $\dot{\mathcal{O}}$ to obtain a strict prefix of $\dot{\mathcal{O}}$, provided this reinforcement yields a valid augmented occurrence net. Intuitively, taking a prefix means reducing the set of possible configurations, which this definition expresses (see also below).

Configuration. An *augmented configuration* κ of $\dot{\mathcal{O}}$ (or *configuration* for short) is a conflict-free prefix of $\dot{\mathcal{O}}$, for $\dot{\#}$. The augmented configuration κ is thus also a configuration of $\mathcal{O}$ enriched with extra causality relations on its events. Obviously, the converse doesn't hold: a configuration of $\mathcal{O}$, may not be transformable into a configuration of $\dot{\mathcal{O}}$, even if causality relations are added, because $\dot{\#}$ is stronger than $\#$.

Morphism. A morphism $\phi : \dot{\mathcal{O}}_1 \rightarrow \dot{\mathcal{O}}_2$ of augmented occurrence nets is a MC net morphism $\phi : \mathcal{O}_1 \rightarrow \mathcal{O}_2$ that partially maps relations $\dot{\prec}_1$ and $\dot{\#}_1$ to $\dot{\prec}_2$ and $\dot{\#}_2$

$$\begin{array}{rll} \forall\, e_1, e_1' \in E_1, & \phi(e_1) \,\dot{\prec}_2\, \phi(e_1') & \Rightarrow\ e_1 \,\dot{\prec}_1\, e_1' \\ \text{and} & \phi(e_1) \,\dot{\#}_2\, \phi(e_1') & \Rightarrow\ e_1 \,\dot{\#}_1\, e_1' \end{array}$$

Notice that causality and conflict relations due to $\rightarrow$ are preserved, only extra relations of $\dot{\prec}_1 \setminus \prec_1$ and $\dot{\#}_1 \setminus \#_1$ can be erased and transformed into concurrency. So morphisms still preserve runs, *i.e.* configurations.

Augmented branching process. The pair $(\dot{\mathcal{O}}, f)$ is an *augmented branching process* (ABP) of net $\mathcal{N}$ iff

1. $f : \mathcal{O} \rightarrow \mathcal{N}$ is a folding of $\mathcal{O}$ into $\mathcal{N}$,
2. if κ and κ' are isomorphic (augmented) configurations of $\dot{\mathcal{O}}$, with identical folding into $\mathcal{N}$, then $\kappa = \kappa'$.

The second requirement generalizes the parsimony criterion appearing in the definition of an ordinary BP, and constrains an ABP to represent at most once a given "augmented" run of $\mathcal{N}$. Notice that the parsimony criterion of ordinary BP doesn't hold here: a given co-set $\mathcal{X}$ of $\dot{\mathcal{O}}$ may be followed by two distinct events e, e' mapped to the same transition of $\mathcal{N}$ by f. This happens when e and e' are not related to the other events of $\dot{\mathcal{O}}$ in the same manner for $\dot{\prec}$, which means that these events have different pasts. This is illustrated in Fig. 4.23. As a consequence, the pair $(\mathcal{O}, f)$ obtained by removing the extra relations, generally violates the parsimony criterion of ordinary BP.

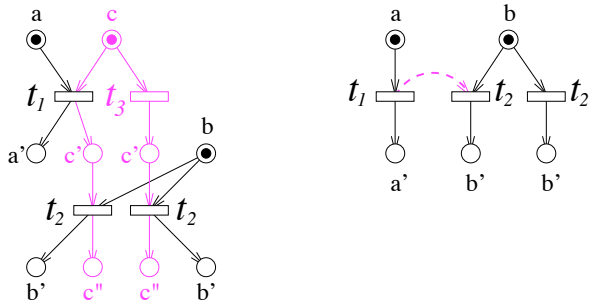


Figure 4.23: *Projection of a BP that removes component $\bar{c}$ (and thus t_3). This yields two runs: one where t_1 precedes t_2, and one where t_1 and t_2 fire concurrently.*

In the sequel, we will make use of *generalized* ABP (GABP) of a net $\mathcal{N}$, which means that point 2 of the definition may not be satisfied. Obviously, when this is not the case, there exists a simple recursive trimming procedure that makes $\dot{\mathcal{O}}$ a true ABP of $\mathcal{N}$. We will denote by $Trim(\dot{\mathcal{O}})$ this operation.

4.5.2 Key property

Augmented branching processes enjoy several interesting properties. For example, removing the extra conflicts in $\dot{\mathcal{O}}$, *i.e.* replacing $\dot{\#}$ by $\#$, yields another ABP of $\mathcal{N}$, that we call the *structure* of $\dot{\mathcal{O}}$. One easily proves that structures have finite width, at any height, which allows us to prove results on ABP by recursion on the height.

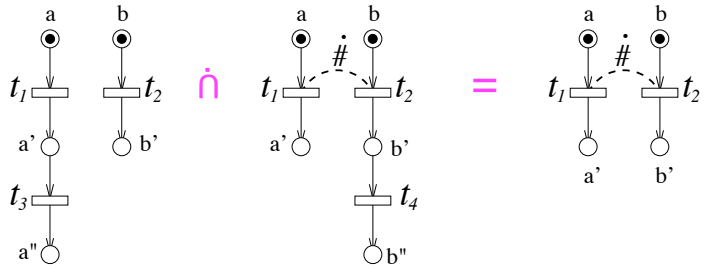


Figure 4.24: *Intersection of two ABP (inherited conflicts are not represented, for clarity).*

Several important properties are related to the ability of ABP to describe sets of runs of $\mathcal{N}$. If every configuration of $\dot{\mathcal{O}}_1$ is isomorphic to a configuration of $\dot{\mathcal{O}}_2$, then

$\dot{\mathcal{O}}_1$ is isomorphic to a prefix of $\dot{\mathcal{O}}_2$. In terms of operations on ABP, one can define the intersection $\dot{\mathcal{O}}_1 \dot{\cap} \dot{\mathcal{O}}_2$ that exactly contains augmented configurations appearing in both $\dot{\mathcal{O}}_i$. It is obtained as the intersection of the structures of the $\dot{\mathcal{O}}_i$, followed by a suitable definition of the extra conflicts (see Fig.4.24). In the same manner, one can also define the union $\dot{\mathcal{O}}_1 \dot{\cup} \dot{\mathcal{O}}_2$, that exactly contains configurations appearing at least in one of the $\dot{\mathcal{O}}_i$ (see Fig.4.25). This strongly contrasts with ordinary branching processes, where the union introduces redundant runs.

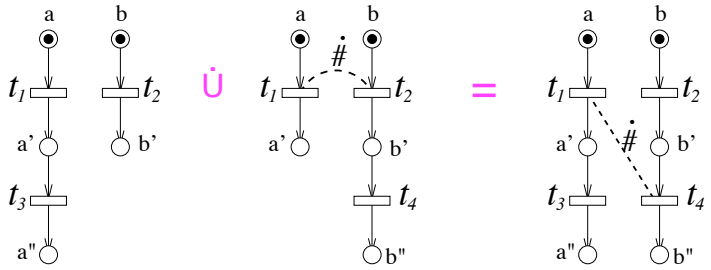


Figure 4.25: *Union of ABP (only minimal conflicts are represented).*

In reality, these properties are consequences of

Proposition 12 *There is a one to one correspondence between prefix-closed sets of (augmented) configurations of $\mathcal{N}$ and augmented branching processes of $\mathcal{N}$.*

So ABP are stronger that ordinary BP in the sense that one is not limited to configuration sets closed by concurrent suffix extension (compare to lemma 8). So unions and intersections of ABPs can be understood as operating on configuration sets.

Proposition 12 deserves one extra comment. We know that distributed computations are possible with configuration sets, just like in the sequential semantics one can compute with sub-languages. As ABP provide a compact encoding of any configuration set (up to prefix closure, which is not a severe limitation), it is likely that one can compute with them. The effort simply consists in proving that computations can be directly performed on ABP, without transforming them into configuration sets.

Since we have the union of ABP, one could define a notion of "augmented unfolding," by taking the union of all augmented configurations of $\mathcal{N}$. This is of little interest, however. First of all because that would yield a huge object, containing all extensions of all (standard) configurations of $\mathcal{N}$. Secondly, this object wouldn't have the universal property, for the definition of morphisms that we took. Consider for example the ABP $\dot{\mathcal{O}}$ in Fig. 4.23, right. The augmented configuration $\dot{\kappa}$ of $\mathcal{N}$ formed by a firing of t_1 followed by a firing of t_2 can be sent into $\dot{\mathcal{O}}$ with two morphisms: one that preserves the causality, and one that erases it.

4.5.3 Operations on ABP: product, pullback, projection

Product. Let $\dot{\mathcal{O}}_1, \dot{\mathcal{O}}_2$ be two AON, we define their product $\dot{\mathcal{O}} = \dot{\mathcal{O}}_1 \times_O \dot{\mathcal{O}}_2$ in the category $\dot{Occ}$ of augmented occurrence nets by extending the recursive procedure

computing the product in *Occ*. This takes the following form, where the $\psi_i : \dot{\mathcal{O}} \to \dot{\mathcal{O}}_i$ as canonical mappings

Procedure 3

- Initialization :
  - $C^0 = C^0_1 \uplus C^0_2$, and the $\psi_i : C^0 \to C^0_i$ follow accordingly,
  - set $C = C^0$ and $E = \emptyset$ ($\rightarrow$, $\dot{\prec}$ and $\dot{\#}$ are empty as well).
- Recursion (until stability) :
  - Let κ be a configuration of $\dot{\mathcal{O}}$ and X an extremal co-set of κ, let $\kappa_i = \psi_i(\kappa)$ and $X_i = \psi_i(X),\ i = 1, 2$
  - if $\exists (e_1, e_2) \in E_1 \times E_2,\ \lambda_1(e_1) = \lambda_2(e_2),\ {}^\bullet e_i = X_i$, and $\nexists e \in E,\ {}^\bullet e = X,\ \psi_i(e) = e_i$,
    - create such e in E,
    - create a set $X' = X'_1 \uplus X'_2$ of $|e_1^\bullet| + |e_2^\bullet|$ new conditions in C, with $e^\bullet = X'$ and $\psi_i(X'_i) = e_i^\bullet$,
    - $\forall e' \in \kappa \cap E$, set $e' \dot{\prec} e$ if $\psi_i(e') \dot{\prec}_i \psi_i(e)$ for some i,
    - $\forall e' \in E$, set $e' \,\dot{\#}\, e$ if $\psi_i(e') \,\dot{\#}_i\, \psi_i(e)$ for some i.
  - if $\exists e_1 \in E_1,\ \lambda_1(e_1) \in \Lambda_1 \setminus \Lambda_2,\ {}^\bullet e_1 = X_1$ and $\nexists e \in E,\ {}^\bullet e = X,\ \psi_1(e) = e_1$
    - create such e in E,
    - create a set X' of $|e_1^\bullet|$ new conditions in C, with $e^\bullet = X'$ and $\psi_i(X') = e_1^\bullet$,
    - $\forall e' \in \kappa \cap E$, set $e' \dot{\prec} e$ if $\psi_i(e') \dot{\prec}_i \psi_i(e)$ for some i,
    - $\forall e' \in E$, set $e' \,\dot{\#}\, e$ if $\psi_i(e') \,\dot{\#}_i\, \psi_i(e)$ for some i.
  - symmetrically for private events of $\dot{\mathcal{O}}_2$.

Of course, $\Lambda = \Lambda_1 \cup \Lambda_2$ and the labeling of events is inherited through the ψ_i, as well as the partitioning ν on conditions. Observe that the recursion does not create all extra causalities and extra conflicts, but only the minimal ones. Therefore, in checking configurations κ for possible extension, one must take into account the causal closure and inheritance of conflicts.

This procedure resembles very much what one would obtain for the product in *Occ*, by coupling the unfolding procedure 2 to the definition of the product in *Nets*. Apart the treatment of $\dot{\prec}$ and $\dot{\#}$, the difference is that events are connected to a *pair* co-set plus configuration instead of a co-set only. This translates the difference of the parsimony criteria of BP compared to ABP.

Proposition 13 *Procedure 3 computes the categorical product in $\dot{Occ}$.*

This entails the associativity of $\times_O$. Proposition 13 has to be proved directly since $\times_O$ can't be derived from the product in *Nets via* a suitable adjunction, this time.

Notice that the product of AON is very close to a product of (labeled) event structures [17, 26, 106, 111], which is consistent with their nature.

Considering augmented branching processes, the picture is not as nice as with ordinary BP, because the parsimony criterion is lost by the product. One has

Proposition 14 *Let* $(\dot{\mathcal{O}}_1, f_1), (\dot{\mathcal{O}}_2, f_2)$ *be ABP of* $\mathcal{N}_1, \mathcal{N}_2$ *respectively. Then* $\dot{\mathcal{O}} = \dot{\mathcal{O}}_1 \times_O \dot{\mathcal{O}}_2$ *provided with the folding* $f = (f_1 \circ \psi_1, f_2 \circ \psi_2)$ *is a* generalized *ABP of* $\mathcal{N} = \mathcal{N}_1 \times \mathcal{N}_2$. *A true ABP of* $\mathcal{N}$ *is obtained by trimming* $\mathcal{O}$.

This phenomenon is illustrated in Fig. 4.26. Fortunately, however, computing with generalized ABP is not so much bothering, except for the size of the objects involved :

Lemma 10 *Let* $\dot{\mathcal{O}}_1, \ldots, \dot{\mathcal{O}}_N$ *be GABP of* $\mathcal{N}_1, \ldots, \mathcal{N}_N$ *resp., then*

$$Trim(\dot{\mathcal{O}}_1 \times_O \ldots \times_O \dot{\mathcal{O}}_N) \quad = \quad Trim[Trim(\dot{\mathcal{O}}_1) \times_O \ldots \times_O Trim(\dot{\mathcal{O}}_N)] \qquad (4.29)$$

For distributed computations on ABP, one can very well include a trimming in the definition of the product. This loses the canonical morphisms from the product to its factors, but doesn't change the results in terms of configuration sets of the underlying nets.

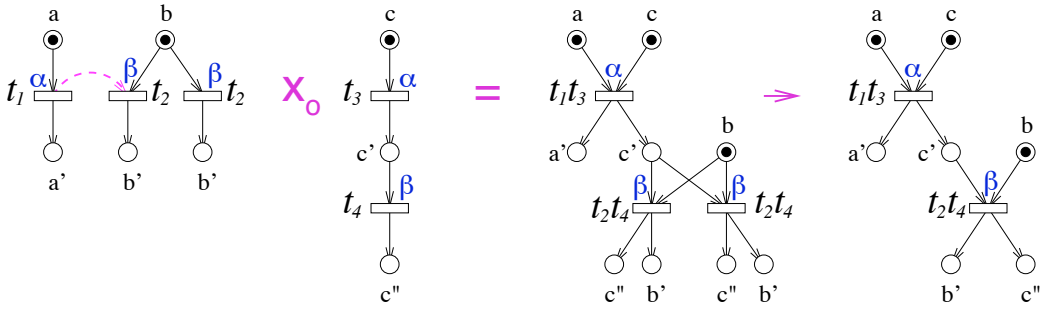


Figure 4.26: *From left to right, two ABP* $\dot{\mathcal{O}}_1, \dot{\mathcal{O}}_2$, *their product* $\dot{\mathcal{O}}_1 \times_O \dot{\mathcal{O}}_2$, *and* $Trim(\dot{\mathcal{O}}_1 \times_O \dot{\mathcal{O}}_2)$.

Pullback. Moving to pullbacks, it is not known whether they all exist in $\dot{Occ}$. But in the simple case where morphisms $\phi_i : \dot{\mathcal{O}}_i \to \dot{\mathcal{O}}_0$ are partial functions, the pullback $\dot{\mathcal{O}}_1 \overset{\dot{\mathcal{O}}_0}{\wedge_O} \dot{\mathcal{O}}_2$ can be obtained with a slight modifications of Procedure 3 above. And it enjoys the same properties as the product. This is the only important case for our computations.

As a remark in passing, notice that the pullback behaves differently on ABP than on BP. For example, if $\mathcal{O}, \mathcal{O}'$ are two ordinary BP of the same net, then $\mathcal{O} \wedge_O \mathcal{O}'$ coincides with $\mathcal{O} \cap \mathcal{O}'$. This is not true anymore with ABP : one has $\dot{\mathcal{O}} \cap \dot{\mathcal{O}}' \sqsubseteq Trim(\dot{\mathcal{O}} \wedge_O \dot{\mathcal{O}}')$.

Minimal product covering. This property doesn't change. Given $\dot{\mathcal{O}} = \dot{\mathcal{O}}_1 \times_O \ldots \times_O \dot{\mathcal{O}}_N$, and $\dot{\mathcal{O}}'_i = \psi_i(\dot{\mathcal{O}})$, one still has $\dot{\mathcal{O}}'_i \sqsubseteq \dot{\mathcal{O}}_i$, $\dot{\mathcal{O}} = \dot{\mathcal{O}}'_1 \times_O \ldots \times_O \dot{\mathcal{O}}'_N$ and the minimality of the $\dot{\mathcal{O}}'_i$. Same thing for the pullback.

Projection. For product ABP $\dot{\mathcal{O}}$ as above (or their prefixes), taking the image $\psi_i(\dot{\mathcal{O}})$ erases causality and conflict relations inherited from the other $\dot{\mathcal{O}}_j$. This was precisely the weakness of computations based on ordinary BP. Therefore we rather define the projection as follows :

$$\dot{\Pi}_i(\dot{\mathcal{O}}) \quad = \quad Trim[\,\dot{\mathcal{O}}_{|Dom(\psi_i)}\,] \tag{4.30}$$

The restriction preserves all inherited causality/conflict relations, while selecting only conditions and events that correspond to component $\mathcal{N}_i$. The relation $\dot{\mathcal{O}} = \dot{\Pi}_1(\dot{\mathcal{O}}) \times_O \ldots \times_O \dot{\Pi}_N(\dot{\mathcal{O}})$ is preserved.

Moreover, for an ordinary BP $\mathcal{O} = \mathcal{O}_1 \times_O \ldots \times_O \mathcal{O}_N$, one easily recovers minimal factors $\mathcal{O}'_i = \Pi_i(\mathcal{O})$ by removing all extra relations in $\dot{\Pi}_i(\mathcal{O})$ and trimming the result. Therefore, one can take as intermediary objective the computation of the minimal product covering in terms of ABP.

4.5.4 Separation theorem

At this point, we have projections $\dot{\Pi}_I$ defined on ABP, that obviously satisfy axioms (a1) and (a2), and a combination by pullback $\wedge$ that satisfies axiom (a4) with the empty ABP as neutral element. Only the central axiom (a3) is missing to enable distributed computations.

Theorem 13 *Let $I, K \subseteq \{1, \ldots, N\}$, and take $J = I \cap K$. Let $\dot{\mathcal{O}}_I, \dot{\mathcal{O}}_K$ be branching processes of $\mathcal{N}_I, \mathcal{N}_K$ respectively, and assume that shared sites $\mathcal{N}_J$ capture all interaction labels between $\mathcal{N}_I$ and $\mathcal{N}_K$,* i.e. *$\Lambda_I \cap \Lambda_K \subseteq \Lambda_J$. Then*

$$\dot{\Pi}_J(\dot{\mathcal{O}}_I \wedge_O \dot{\mathcal{O}}_K) \quad = \quad Trim[\,\Pi_J(\dot{\mathcal{O}}_I) \wedge_O \Pi_J(\dot{\mathcal{O}}_K)\,] \tag{4.31}$$

This relation resembles more axiom (a3) if the trimming is introduced into $\wedge_O$ to form $\dot{\wedge}_O$. (4.31) becomes $\dot{\Pi}_J(\dot{\mathcal{O}}_I \dot{\wedge}_O \dot{\mathcal{O}}_K) = \Pi_J(\dot{\mathcal{O}}_I) \dot{\wedge}_O \Pi_J(\dot{\mathcal{O}}_K)$. Notice that when the separation criterion is violates, one still has $\dot{\Pi}_J(\dot{\mathcal{O}}_I \dot{\wedge}_O \dot{\mathcal{O}}_K) \sqsubseteq \Pi_J(\dot{\mathcal{O}}_I) \dot{\wedge}_O \Pi_J(\dot{\mathcal{O}}_K)$.

With theorem 13, we are thus fully equipped to perform the distributed computations of the previous section, without the assumption of non-misleading projections.

4.5.5 Weak involutivity

As mentioned section 4.4.5 (see also sections 3.6.3 and 2.2.4), when a system $\mathcal{N} = \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_M$ doesn't live on a tree, message passing algorithms (MPA) still provide good approximations of the reduced local diagnoses, provided computations are performed on involutive objects. This is the case of branching processes :

$$\mathcal{O}_I \wedge \Pi_J(\mathcal{O}_I) \quad = \quad \mathcal{O}_I \tag{4.32}$$

which corresponds to axiom (a8) section 2.2.4, so all MPA have a unique and identical stationary point. The latter is formed of partially reduced local diagnoses, that form a (non minimal) pullback covering of the global diagnosis.

What about computations based on augmented branching processes ? Unfortunately, involutivity is lost, only a weak form remains : For $\dot{\mathcal{O}}_I$ and ABP of $\mathcal{N}_I$, one has

$$\dot{\mathcal{O}}_I \wedge \dot{\Pi}_J(\dot{\mathcal{O}}_I) \quad \sqsupseteq \quad \dot{\mathcal{O}}_I \tag{4.33}$$

So composing an ABP with part of itself generally introduces extra (reinforced) configurations. This doesn't mean however that turbo procedures are excluded for ABP. Let us define the relation $\Subset$ on GABP of a net $\mathcal{N}$ by

$$\dot{\mathcal{O}}_1 \Subset \dot{\mathcal{O}}_2 \quad \Leftrightarrow \quad \dot{\mathcal{O}}_1 \sqsubseteq \dot{\mathcal{O}}_1 \wedge \dot{\mathcal{O}}_2 \tag{4.34}$$

Proposition 15 *The pre-order $\Subset$ on GABPs of $\mathcal{N}$, satisfies axioms (a5,a6,a7).*

So we are almost in an involutive setting. Let us define the equivalence relation $\equiv$ on (G)ABP of $\mathcal{N}$ by :

$$\dot{\mathcal{O}}_1 \equiv \dot{\mathcal{O}}_2 \quad \Leftrightarrow \quad \dot{\mathcal{O}}_1 \Subset \dot{\mathcal{O}}_2 \Subset \dot{\mathcal{O}}_1 \tag{4.35}$$

Then the involutivity is recovered provided one replaces $=$ by $\equiv$, *i.e.* (4.33) becomes $\dot{\mathcal{O}}_I \wedge \dot{\Pi}_J(\dot{\mathcal{O}}_I) \equiv \dot{\mathcal{O}}_I$. So turbo algorithms based on ABP converge to a unique stationary point, defined in terms of equivalence classes of $\equiv$ (see theorems 2 and 4). These stationary classes are identical for all algorithms, but the latter may however reach different ABP in these classes.

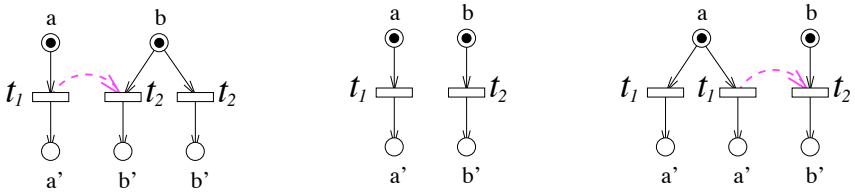


Figure 4.27: *Three equivalent ABP of the same net.*

To help intuition, Fig. 4.27 depicts equivalent ABP of the same net. In substance, two GABP $\dot{\mathcal{O}}_1, \dot{\mathcal{O}}_2$ are equivalent if configurations of $\dot{\mathcal{O}}_1$ are obtained by reinforcing configurations of $\dot{\mathcal{O}}_2$ with extra causal links, and *vice-versa.* Or equivalently if each GABP can be folded into the other.

Proposition 16 *Equivalence classes of GABP are stable by $\wedge$, by trimming, by intersection and by union of GABP. So they admit a minimal element.*

In Fig. 4.27, the minimal net is the central one : the configurations of the others are obtained by reinforcing configurations of this minimal GABP.

By axioms (a5,a6,a7), we know that the equivalence of GABP is preserved by pullback $\dot{\wedge}$ (which includes trimming), and by projection $\dot{\Pi}$. But these operations don't preserve the minimality. Nevertheless, if all computations are followed by a minimization step, then involutivity in the strict sense is recovered.

In conclusion, one can indeed run turbo algorithms on ABP, and get the same properties as before, up to the minimality of the result. Which is not so much bothering since in the end extra conflict and causality relations are discarded to get the partially reduced local diagnoses.

4.6 Summary

This chapter is the equivalent of the previous one for the true concurrency semantics. We have introduced a second way of composing automata, in order to shape the result as a safe Petri net and reveal the concurrency of transitions. Runs of these networks of automata were defined as partial orders of events, or configurations. As for the sequential semantics, sets of configurations can be used as such, or encoded into a more compact data structure called a branching process. One can go further in terms of compactness and define the counterpart of trellis processes for the true concurrency semantics (this is examined in the next chapter).

Concerning the distributed diagnosis problem, computations based on branching processes have the same algebra as in chapter 3 : since projections on labels are impossible, one is again constrained to use projections on sites to transmit information between components, and to use pullbacks to combine information. The main difference lies in the nature of projections : using a naive definition allows to recycle exactly the approach of chapter 3, but this doesn't capture all cases. In particular, this strategy fails when components share an interface that enables concurrent events, which may lead to "misleading projections." To capture the general case, we have introduced the notion of augmented branching process, that combines the definitions of BP and of event structures. Their properties give them the flexibility of configuration sets, and allow us to recover a clean definition of projection. Up to this little modification, the rest of the algebra can be recovered. A slight difference appears in the involutivity property, that takes a weaker form, but still sufficiently strong to prove the convergence of turbo algorithms on cyclic networks of automata.

The nature of augmented branching processes, that resemble very much event structures, suggests that one could imagine working directly with event structures (see [41] for a first solution). A crucial point would be to define projections of these objects on label sets, which would bring us back to the simple case of the "direct graph" setting (see the language approach in section 3.4), and would avoid the burden of using pullbacks. This is still a research issue.

Related work

The unfolding technique was introduced in the early 80's [86, 110, 111] as a convenient way to represent runs of concurrent systems, in the so-called true concurrency semantics. The expression "true concurrency" refers to the fact that runs are represented as partial orders of events, by contrast with other models of concurrency like Mazurkiewicz traces [27]. A trace is an equivalence class of sequences of events, where the equivalence is defined by the permutation of consecutive independent events.

Unfoldings were revisited in the 90's as a convenient tool for verification purposes [31, 82]. To this end, a crucial step is the construction of a finite complete prefix of the unfolding, *i.e.* a finite prefix that is sufficient to check a given property on the system. The most frequent definition of completeness refers to prefixes that contain all possible markings of the underlying net. Mac Millan proposed the first construction [83], based on the key notion of adequate order (to compare configu-

rations). It was later refined by Esparza *et al.* [33, 34, 35] with the introduction of smarter adequate orders. It can then be proved that a finite complete prefix is not larger than the marking graph of the underlying net. These approaches were algorithmic : the prefix was the output of a procedure. Khomenko [64, 67] introduced the idea of a canonical prefix, based on the notion of cutting context that defines *a priori* at which events one should cut the unfolding. He also proposed efficient unfolding algorithms, making use of SAT solvers to find co-sets and possible extensions [60, 66]. Among various applications of unfoldings in model-checking, several contributions consider reachability analysis [32, 36, 82, 83] and deadlock detection [65, 85].

The unfolding technique has been applied to various kinds of concurrent systems. Not only safe nets, but also semi-weighted nets [31], products of transition systems (as here) [35] or products of symmetrical nets [23]. More exotic concurrent models have also been explored. For example the important class of Petri nets with read arcs [7, 107], symbolic (or high-level) nets [19], networks of timed automata and time Petri nets [16, 18], or graph grammars. Chatain, in particular, proposed an important improvement with the idea of symbolic unfolding [19], see section 7.2.1. We jointly proved that symbolic unfoldings enjoy the same factorization properties as standard unfoldings, which makes them natural candidates for distributed diagnosis.

Beside their intensive use in model-checking, partial order methods and unfoldings have also been explored in the discrete event systems community, but apparently with a lower impact. Let us mention works in liveness enforcement [59], and several interesting contributions in controller synthesis [51, 52, 53]. In distributed diagnosis applications, partial order methods have been used for state reconstruction in the case of components interacting by places [14, 62, 63], with the interesting idea of backward unfolding. They have also been used, in parallel to our own work, for communicating automata [88]. The algebraization of distributed computations presented in this chapter is original [9, 41]. The early versions of this approach have much benefited from the work of Winskel [109, 111, 112], and in particular [110], that enabled a clean and concise re-expression of all our results. Winskel introduced a simple and elegant way of deriving factorization results on unfoldings, by means of category theory, that we use in different places of this document. So far, it seems that factorization aspects have not been exploited for modular model-checking applications.

Chapter 5

Trellis unfolding for concurrent systems

The trellis of an automaton represents all its possible runs in a compact structure, with a complexity that is typically linear in time (=length of runs). The idea is to merge not only the common pasts, but also the common futures of two runs that reach identical states at a the same time. Although such structures are familiar in communities dealing with Markov chains (control, digital communications, etc.), they are apparently new in the field of distributed processings, essentially because of the unnatural multi-clock ingredient that we had to introduce (chapter 4).

This chapter presents the generalization of this idea to true concurrency semantics, which has opened a new and promising research direction. Traditional unfoldings can be seen as operating a *double expansion* of a concurrent system. Time, of course, is unrolled (unfoldings are partial orders of events), but also *conflicts* are expanded : each time there is a choice between n possible events, n branches are created, that will never meet each other again, since conflicts are inherited in an unfolding. By contrast, the notion of trellis is meant to unfold time, but *not conflicts*, and thus results in a more compact structure. Can such a simple idea be applied to runs defined as partial orders ? Surprisingly, the answer is positive. And as a nice feature, factorization properties can again be proved, provided merge points are defined according to a vector clock, as in chapter 3. In reality, the algebraic similarities are extremely close, which allows us to replicate our approach to distributed diagnosis, up to slight modifications of the key theorems.

The chapter is organized as follows. We first define trellis processes, and study their factorization properties. We then examine their relations to unfoldings. Finally, we show that they admit a natural notion of projection, for which the separation theorem can again be established. This makes trellis processes suitable for distributed diagnosis applications, which we illustrate on an example.

5.1 Trellis nets

This section defines the category of trellis nets, a family of nets where time is unfolded, but not necessarily conflicts. This category is thus intermediate between

Occ, occurrence nets, and *Nets*, safe Petri nets. We are still in the multi-clock setting, that equips nets with a partition of places, used to identify components.

5.1.1 Definition

Pre-trellis net. The MC net $\mathcal{T} = (C, E, \rightarrow, C^0, \nu)$ is a *pre-trellis net* iff it satisfies:

1. $C^0 = \{c \in C \,:\, {}^\bullet c = \emptyset\}$,
2. for every $c \in C^0$, the automaton $\mathcal{T}_{|\bar{c}}$ has no circuit (*i.e.* its flow relation defines a partial order).

The definition of pre-trellis nets is much less restrictive than the definition of occurrence nets. Specifically, point 1 is preserved, point 2 is weakened since $\rightarrow^*$ is not any more required to define a partial order, and we have abandoned points 3 and 4: conflicting branches are now allowed to merge on conditions.

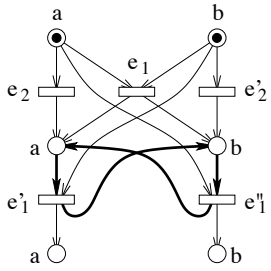


Figure 5.1: *A pre-trellis net containing a circuit (thick arrows).*

As an oriented graph, and by contrast with occurrence nets, a pre-trellis net is not necessarily a partial order. Fig. 5.1 gives a counter-example of a pre-trellis net containing a circuit. However, one has the following property:

Lemma 11 *No run of a pre-trellis net $\mathcal{T}$ can have a loop,* i.e. *can fill twice the same place. As a consequence, the restriction $\mathcal{T}_{|\sigma}$ of $\mathcal{T}$ to (nodes involved in) any run σ defines a partial order of nodes.*

Therefore it makes sense to express runs of $\mathcal{T}$ as configurations κ rather than sequences σ of transitions.

Configuration, trellis net. In an occurrence net, every event belongs at least to one configuration, and so is reachable. This is not guaranteed anymore in a pre-trellis net (see $\mathcal{T}_1$ in fig. 5.3), so we must refine our definition. We define a *configuration* κ of a pre-trellis net $\mathcal{T} = (C, E, \rightarrow, C^0, \nu)$ as a sub-net of $\mathcal{T}$ satisfying

1. $C^0 \subseteq \kappa$,
2. $\forall e \in E \cap \kappa$, ${}^\bullet e \subseteq \kappa$ and $e^\bullet \subseteq \kappa$: each event has all its causes and consequences,
3. $\forall c \in C \cap \kappa$, $|{}^\bullet c \cap \kappa| = 1$ or $c \in C^0$: each condition is either minimal or has one of its possible causes,

4. $\forall c \in C \cap \kappa,\ |c^\bullet \cap \kappa| \leq 1$: each condition triggers at most one event,

5. the restriction of $\mathcal{T}$ to nodes of κ is a partial order.

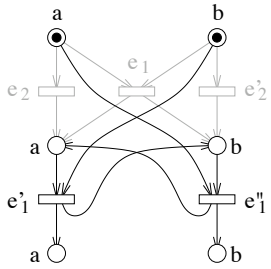


Figure 5.2: *In the net of fig. 5.1, a subset of nodes satisfying the first four requirements of a configuration, but failing on the last one.*

This definition is close to the one introduced for ONs, apart from the fact that $|^\bullet c| \leq 1$ is not automatic anymore in a pre-trellis net. So one must not only solve conflicts forward (point 4) but also backwards (point 3), to get a valid conflict-free ON. And the requirement that a configuration is "causally closed" is now spread on 2, 3 and 4. The last point is suggested by lemma 11, and is indeed necessary since points 1 to 4 alone do not guarantee this property (see a counter-example in fig. 5.2). With the above definition, it is straightforward to check that a sequence σ is a run of $\mathcal{T}$ iff it corresponds to a linear extension of some configuration κ of $\mathcal{T}$. And so an event of a pre-trellis net is reachable iff it belongs to a configuration.

We thus define a *trellis net* (TN) as a pre-trellis net where each event belongs at least to one finite configuration (see fig. 5.3 for examples).

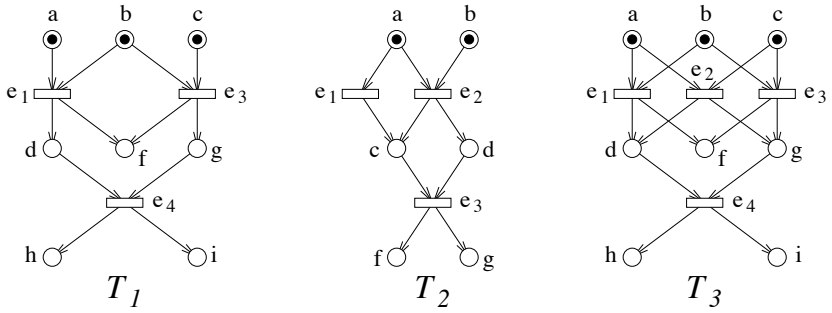


Figure 5.3: *$\mathcal{T}_1$ is a pre-trellis net but not a trellis net: event e_4 is unreachable. The other nets are trellis nets: all events are reachable. In $\mathcal{T}_2$, e_1 and e_3 are not causally related... but in conflict ! $\mathcal{T}_3$ displays a non binary conflict: $\{d, f\}$, $\{f, g\}$ and $\{d, g\}$ are all pairs of concurrent conditions, but the triple $\{d, f, g\}$ appears in no run. Removing e_2 in $\mathcal{T}_3$ doesn't yield a valid prefix: we are back to $\mathcal{T}_1$ which is not a trellis net.*

Concurrency and conflict. From the definitions above, one sees that both ONs and TNs are graphical structures encoding families of configurations in different

ways. TNs offer the advantage of being more compact... at the expense of a more complex display of configurations. In particular, the familiar causality, conflict and concurrency relations on events do not have any more a simple graphical translation (see $\mathcal{T}_2$ in fig. 5.3). This is due to the fact that, in a TP, an event (or a condition) generally appears on top of several histories. This phenomenon introduces a strong contrast with ONs, where a node belongs to a unique minimal configuration, its causal closure. As a consequence, concurrency and causality are now "context dependent:" two events may be concurrent in one configuration, and appear as causally related in another (fig. 5.4).

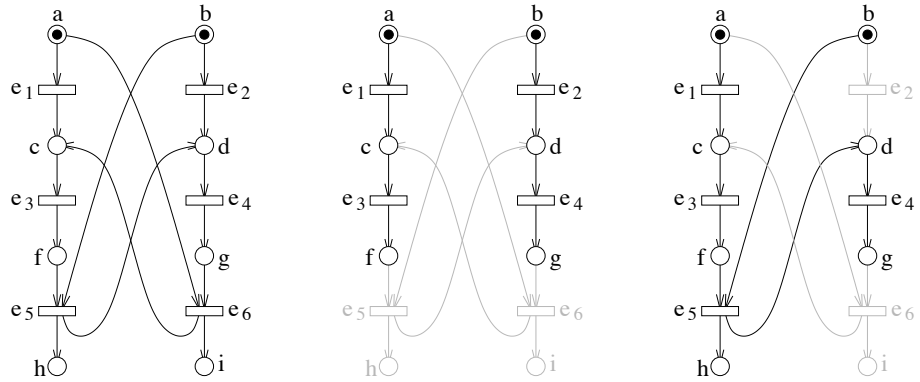


Figure 5.4: *On this trellis net (left), events e_3 and e_4 appear in several configurations. They can be concurrent in one of them (center) and causally related in another (right).*

It is important to define co-sets in a trellis net, *i.e.* to determine conditions that can be used at the same time to connect one more event to the structure. To define this extended notion of concurrency, we thus have to abstract the context. Let $x_1, x_2, \ldots, x_n$ be n nodes of $\mathcal{T}$, they are *concurrent* in $\mathcal{T}$, denoted by $\perp\!\!\!\perp(x_1, x_2, \ldots, x_n)$, iff there exists a configuration κ where they appear as concurrent nodes. In the example of fig. 5.4 (left), e_3 and e_4 are thus declared concurrent for this extended notion. The notion of co-set (of conditions) derives from this definition. Observe that in a TN, concurrency can no longer be derived from pairwise relations, by contrast with ONs (see $\mathcal{T}_3$ in fig. 5.3). In the same way, an extended notion of conflict can be defined as follows: $x_1, x_2, \ldots, x_n$ are in conflict, $\#(x_1, x_2, \ldots, x_n)$, iff there is no configuration containing all of them (for example $\#(e_5, e_6)$ in fig. 5.4). Again, # cannot be derived from pairwise relations, *i.e.* conflict is not binary in TNs.

Prefix. Prefixes are less easy to define graphically for TNs than for ONs. Let $\mathcal{T}$ be a TN, $\mathcal{T}'$ is a *prefix* of $\mathcal{T}$ ($\mathcal{T}' \sqsubseteq \mathcal{T}$) iff

1. $\mathcal{T}'$ is a sub-net of $\mathcal{T}$,
2. $\{c \text{ condition of } \mathcal{T}, {}^\bullet c = \emptyset\} = \{c' \text{ condition of } \mathcal{T}', {}^\bullet c' = \emptyset\}$
3. $\forall e$ event of $\mathcal{T}$, $e \in \mathcal{T}' \Rightarrow [{}^\bullet e \subseteq \mathcal{T}' \text{ and } e^\bullet \subseteq \mathcal{T}']$,

4. $\mathcal{T}'$ is a trellis net.

The last requirement imposes that every event in the sub-net $\mathcal{T}'$ remains reachable. To illustrate its necessity, consider $\mathcal{T}_3$ in fig. 5.3 : if e_2 is removed, points 1-3 are satisfied, but e_4 becomes unreachable. Of course, $\sqsubseteq$ on TNs extends the relation $\sqsubseteq$ on ONs. Notice also that $\mathcal{T}' \sqsubseteq \mathcal{T}$ implies the existence of an injective morphism $\phi : \mathcal{T}' \to \mathcal{T}$ (which means here that ϕ is a total function).

Height function. The definition of trellis nets now allows us to merge conflicting conditions reached by different runs. However, this leaves a large amount of flexibility. But if one wishes to get a universal object to represent configuration sets, some kind of guideline is necessary to indicate where merges must be performed.

Let us define a *string* as a configuration $\sigma = (C, E, \to, C^0, \nu)$ in Occ where $|C^0| = 1$. So σ has a single class and thus corresponds to a sequence alternating conditions and events. The *height* $H_\sigma(c)$ of condition c in σ is given by

$$H_\sigma(c) \quad = \quad |\{c' \in C,\ c' \to^* c\}| \tag{5.1}$$

In a general configuration κ, we define the height of a condition c by focusing on the component $\kappa_{|\bar{c}}$ that contains c, so we set $H_\kappa(c) \triangleq H_\sigma(c)$ where $\sigma = \kappa_{|\bar{c}}$, or equivalently

$$H_\kappa(c) \quad = \quad |\{c' \in C,\ \nu(c') = \nu(c),\ c' \to^* c\}| \tag{5.2}$$

A trellis net $\mathcal{T}$ is *correctly folded* for H iff for every condition c of $\mathcal{T}$ and every pair of strings σ, σ' containing c in $\mathcal{T}_{|\bar{c}}$, one has $H_\sigma(c) = H_{\sigma'}(c)$. Equivalently, $\mathcal{T}$ is correctly folded iff $\forall c \in C,\ \forall \kappa, \kappa'$ configurations of $\mathcal{T}$ containing c, $H_\kappa(c) = H_{\kappa'}(c)$. We denote this common value by $H_\mathcal{T}(c)$ or simply $H(c)$ when there is no ambiguity. Fig. 5.5 illustrates this property.

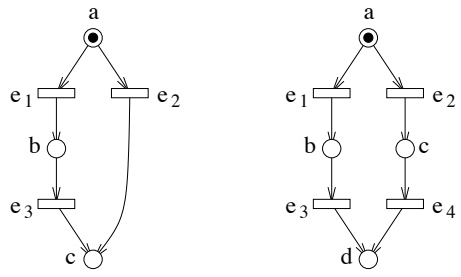


Figure 5.5: *Two trellis nets ; the left one is not H-compliant, the other one is.*

Category of trellis nets. In the sequel, we only consider H-compliant labeled trellis nets. The latter, associated to the usual notion of morphism (of MCNs), form the category Tr. So we have three nested categories : $Occ \subset Tr \subset Nets$.

5.1.2 Trellis process and time unfolding of a net

Trellis process. Reproducing the developments around occurrence nets, trellis nets can be used to represent runs of a given LMC net $\mathcal{N}$. Let $\mathcal{T} = (C, E, \rightarrow, C^0, \nu, \lambda, \Lambda)$ be a labeled trellis net and $f : \mathcal{T} \rightarrow \mathcal{N}$ a morphism, the pair $(\mathcal{T}, f)$ forms a *trellis process* (TP) of $\mathcal{N}$ iff

1. f is a folding of $\mathcal{T}$ (*i.e.* a total function on $\mathcal{T}$),
2. $\mathcal{T}$ is a parsimonious description of runs of $\mathcal{N}$:

$$\forall e, e' \in E,\ \ [{}^\bullet e = {}^\bullet e',\ f(e) = f(e')] \ \Rightarrow\ e = e'$$

3. $\mathcal{T}$ is maximally folded :

$$\forall c, c' \in C,\ \ [H(c) = H(c'),\ f(c) = f(c')] \ \Rightarrow\ c = c'$$

The novelty with respect to branching processes is thus the merge imposed by 3, that operates as an extra parsimony criterion to describe runs of $\mathcal{N}$.

As f is a folding of $\mathcal{T}$ into $\mathcal{N}$, every configuration κ of $\mathcal{T}$ represents a run of $\mathcal{N}$ in the true concurrency semantics, and has a counterpart in $\mathcal{U}_\mathcal{N}$. So a trellis process of $\mathcal{N}$ corresponds to a collection of runs of $\mathcal{N}$. Conversely, a run of $\mathcal{N}$ is represented by at most one configuration in $\mathcal{T}$: If κ_1 and κ_2 are isomorphic and folded into $\mathcal{N}$ in the same way, then they are identical. Indeed, one has $H_{\kappa_1} = H_{\kappa_2}$ which shows that conditions are identical (point 3), from which events are also identical (point 2). As before, we will often omit mentioning the folding f when there's no ambiguity.

Observe the close similarity between the definition above and section 3.5.2 that defines *sequential* trellis processes. Actually, most properties of sequential TP can be transported to the partial order semantics : a TP is isomorphic to the union of its configurations, two TP that have the same configurations are isomorphic, etc.

Time unfolding of a net. One can easily build any trellis process of a net $\mathcal{N} = (P, T, \rightarrow, P^0, \nu_\mathcal{N}, \lambda_\mathcal{N}, \Lambda)$ with a simple recursion, yielding both $\mathcal{T} = (C, E, \rightarrow, C^0, \nu, \lambda, \Lambda)$ and the folding $f : \mathcal{T} \rightarrow \mathcal{N}$. This recursion is a simple refinement of procedure 2, designed for branching processes. A single new feature appears : the merge of newly created conditions once a new event has been connected, in order to satisfy the last requirement of the definition.

Procedure 4

- Initialization :
  - Create $|P^0|$ conditions in C^0, and define a bijection $f : C^0 \rightarrow P^0$.
  - Set $C = C^0$, $E = \emptyset$ and $\rightarrow = \emptyset$.
- Recursion :
  - Let X be a co-set of C and $t \in T$ a transition of $\mathcal{N}$ such that $f(X) = {}^\bullet t$.

– If there doesn't exist an event e in E with ${}^\bullet e = X$ and $f(e) = t$,
  * create a new event e in E with ${}^\bullet e = X$ and $f(e) = t$,
  * create a subset Y of $|t^\bullet|$ new conditions in C, with $Y = e^\bullet$, extend f to have $f : Y \to t^\bullet$ bijective,
  * then, for every $c \in Y$, if $\exists c' \in C,\ f(c') = f(c)$ and $H(c') = H(c)$ then merge c and c'.

The partitioning of conditions $\nu : C \to C^0$ and the labeling of events $\lambda : E \to \Lambda$ are of course inherited from those of $\mathcal{N}$ through f.

As for unfoldings, one can modify procedure 4 to compute the union of an arbitrary number of TP of $\mathcal{N}$. Taking the union of all TP of $\mathcal{N}$ results in a unique maximal TP, that has all the others as prefixes. It is of course obtained as the unique stationary point of procedure 4.

Theorem 14 *Let $\mathcal{N}$ be a multi-clock net, there exists a unique maximal trellis process of $\mathcal{N}$ for the prefix relation. We call it the* trellis *of $\mathcal{N}$ or the* time unfolding *of $\mathcal{N}$, and denote it by $\mathcal{U}^t_{\mathcal{N}}$, with corresponding folding $f^t_{\mathcal{N}} : \mathcal{U}^t_{\mathcal{N}} \to \mathcal{N}$.*

$\mathcal{U}(\mathcal{N})$, the unfolding of $\mathcal{N}$, and $\mathcal{U}^t(\mathcal{N})$, the time unfolding of $\mathcal{N}$, are different encodings for the same configuration set, formed by all trajectories of $\mathcal{N}$.

Fig. 5.6 illustrates the (beginning of the) time unfolding of our running example $\mathcal{N}$. Despite the apparent loop back, that may look surprising, there is no executable circuit in this net. Observe also that at each "level" one more possibility to fire t_1 appears. This is due to the "bypass" transition t_2, that allows us to use a condition g arbitrarily far in the past.

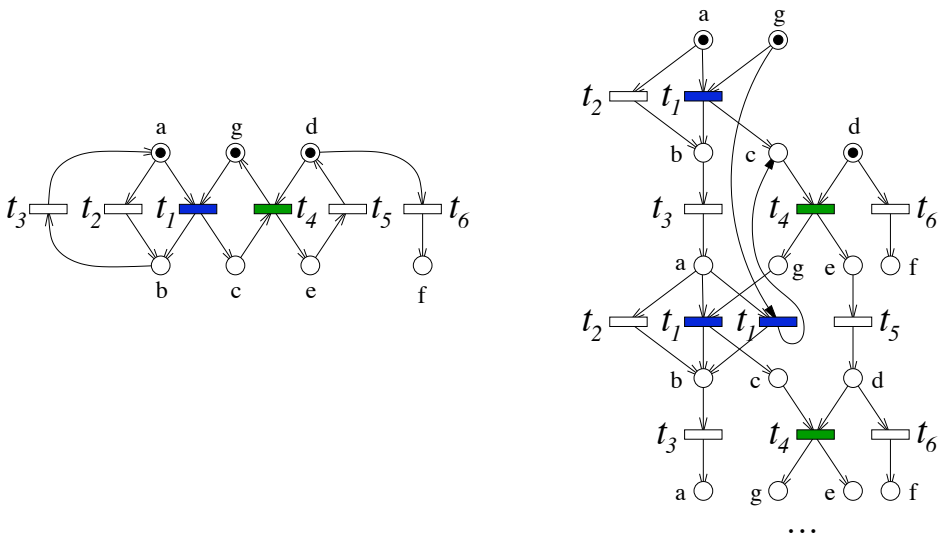


Figure 5.6: *A net (left) and the beginning of its time unfolding (right).*

Expressive power of trellis processes. Like all other structures we have introduced, TP can't encode any configuration set. In fact, the more properties we ask to these structures, the less flexibility they offer. Nevertheless, the sub-languages of $\mathcal{N}$ they describe are sufficient to perform distributed computations.

Let (κ_1, f_1) and (κ_2, f_2) be two configurations of $\mathcal{N}$, leading respectively to cuts C_i, *i.e.* $C_i^0[\kappa_i\rangle C_i$. κ_1 and κ_2 are said to be H-equivalent, denoted by $\kappa_1 \simeq^H \kappa_2$, iff $f_1(C_1) = f_2(C_2)$ and H_{κ_1} coincides with H_{κ_2} on their terminal cut:

$$\forall (c_1, c_2) \in C_1 \times C_2, \quad f_1(c_1) = f_2(c_2) \quad \Rightarrow \quad H_{\kappa_1}(c_1) = H_{\kappa_2}(c_2) \tag{5.3}$$

In other words, κ_1 and κ_2 would finish at the same cut in $\mathcal{U}^t(\mathcal{N})$. A sub-language $\mathcal{L}$ of $\mathcal{N}$, as a configuration set, is suffix-closed iff

$$\forall \kappa = \kappa_1\kappa_1' \in \mathcal{L},\ \forall \kappa_2 \in \mathcal{L}, \quad \kappa_1 \simeq^H \kappa_2 \quad \Rightarrow \quad \kappa_2\kappa_1' \in \mathcal{L} \tag{5.4}$$

Lemma 12 *A sub-language $\mathcal{L}$ of $\mathcal{N}$ can be encoded as a trellis process $(\mathcal{T}, f)$ of $\mathcal{N}$ iff it is prefix- and suffix-closed.*

As before, one can relax a little the prefix-closure necessity by introducing a stop function, that assigns a zero-one value to cuts of a TP.

5.1.3 Factorization properties

Theorem 15 (Universal property of $\mathcal{U}^t_{\mathcal{N}}$) *Let $\mathcal{N}$ be an LMC net, for every trellis net $\mathcal{T}$ in Tr and morphism $\phi : \mathcal{T} \to \mathcal{N}$, there exists a unique morphism $\psi : \mathcal{T} \to \mathcal{U}^t_{\mathcal{N}}$ such that $\phi = f^t_{\mathcal{N}} \circ \psi$.*

By arguments we have already detailed several times, this is sufficient to establish a co-reflection of Tr into $Nets$. The adjoint pair of functors relating these two categories are (F, G) where $F = \subseteq : Tr \to Nets$ is the inclusion functor, and in the reverse direction $G = \mathcal{U}^t : Nets \to Tr$ is simply the time-unfolding operation.

An important property one expects from trellis processes concerns their factorization, since it forms the first pillar of modular computations. As soon as the co-reflection above is established, one mechanically gets the preservation of limits bu $\mathcal{U}^t$, and thus the preservation of products and pullbacks. So one has

$$\mathcal{U}^t(\mathcal{N}_1 \times \mathcal{N}_2) \quad = \quad \mathcal{U}^t(\mathcal{N}_1) \times_T \mathcal{U}^t(\mathcal{N}_2) \tag{5.5}$$

which also proves the existence of a product $\times_T$ in Tr for trellis nets. The latter can actually be *defined* by

$$\mathcal{T}_1 \times_T \mathcal{T}_2 \quad \cong \quad \mathcal{U}^t(\mathcal{T}_1) \times_T \mathcal{U}^t(\mathcal{T}_2) \quad = \quad \mathcal{U}^t(\mathcal{T}_1 \times \mathcal{T}_2) \tag{5.6}$$

Once again, this relation is important in practice: by inserting the definition of product $\times$ on nets into the time unfolding procedure, one gets an effective algorithm to compute recursively products like $\mathcal{T}_1 \times_T \mathcal{T}_2$.

Naturally, the two relations above remain true if one replaces products by pullbacks, which shows the interest of these abstract algebraic approaches.

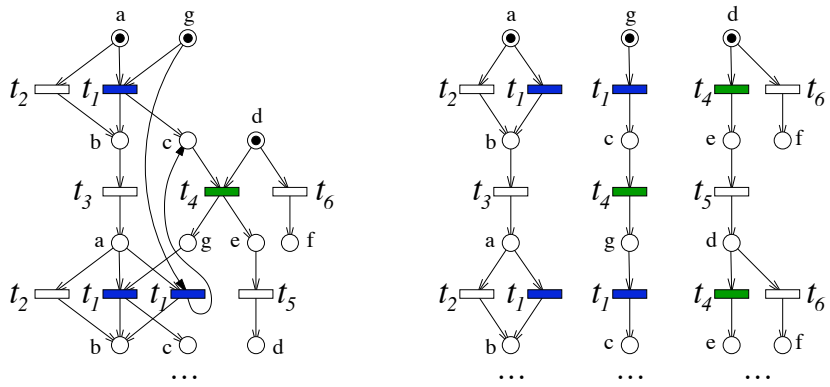


Figure 5.7: *Left : Trellis of the net $\mathcal{N}$ depicted in fig. 5.6. Right : Trellises of its components $\mathcal{N}_{|\bar{a}}, \mathcal{N}_{|\bar{g}}, \mathcal{N}_{|\bar{d}}$. The trellis on the LHS is the product of the three other trellises, in the sense of $\lambda\overline{Tr}$: $\mathcal{U}^t_{\mathcal{N}} = \mathcal{U}^t_{\mathcal{N}_{|\bar{a}}} \times_{\lambda\overline{T}} \mathcal{U}^t_{\mathcal{N}_{|\bar{g}}} \times_{\lambda\overline{T}} \mathcal{U}^t_{\mathcal{N}_{|\bar{d}}}$.*

Fig. 5.7 illustrates the factorization property. The net $\mathcal{N}$ in Fig. 5.6 contains three elementary components $\mathcal{N}_{|\bar{a}}, \mathcal{N}_{|\bar{g}}, \mathcal{N}_{|\bar{d}}$, or sequential machines. Taking this decomposition for $\mathcal{N}$ shows that its trellis $\mathcal{U}^t_{\mathcal{N}}$ is the product of the trellises of these components, which are nothing more than "ordinary" trellises, in the usual sense of that word for automata. The figure illustrates also the interest of factorized representations, that we have already underlined in the previous chapters : While $\mathcal{U}^t_{\mathcal{N}}$ grows in width and complexity, because of the extra t_1 that appears at each level, its factors have a linear complexity in time.

5.2 Relations to unfoldings

5.2.1 Variations around the height function

As for the sequential semantics, the definition of a height function is quite flexible, provided one preserves its vector nature.

Consider the triple $(\mathcal{E}, \odot, \epsilon)$ formed by a set $\mathcal{E}$, an internal composition law $\odot$ in $\mathcal{E}$ and a neutral element ϵ. Let us attach to each LMC net $\mathcal{N}$ a tuple of functions h associating values of $\mathcal{E}$ to the transitions of $\mathcal{N}$. Specifically, h is formed by the vector $(h_p)_{p\in P^0}$ and $h_p : T \to \mathcal{E}$ satisfies $h_p(t) = \epsilon$ whenever $p \notin \nu(^\bullet t)$. In other words, when t doesn't influence the component $\mathcal{N}_{|\bar{p}}$, its value $h_p(t)$ is neutral. The height values attached to a transition can depend on their label, for example, and we consider net morphisms that preserve these h values.

To generalize trellis processes, we simply combine the height values of events to define a height function on conditions. To a condition c of configuration κ, we associate the height $H_\kappa(c)$ by considering the string $\sigma = \kappa_{|\bar{c}}$ that contains c. This string takes the form $c_0 \to e_1 \to c_1 \to e_2 \to \ldots \to e_n \to c \to e_{n+1} \to \ldots$ and we take $H_\kappa(c) = h_{c_0}(e_1) \odot \ldots \odot h_{c_0}(e_n)$. Once a height function is defined, the theory remains unchanged with correctly and maximally folded trellis nets[1].

[1]At this point, this statement is only a conjecture that has not been rigorously proved.

This degree of freedom allows us to build trellises that simply ignore some events in their counting, for example events that don't produce visible labels. So one can unfold time only when observations are produced, and preserve silent cycles unchanged.

On the contrary, one can define a height function that never allows merges in some components. For example, in a configuration (κ, f) of $\mathcal{N}$ consider $h_c(e) = f(e)$, the name of the transition $t = f(e)$ fired in event e, and take for $\odot$ the concatenation of transition names. The trellis of each component $\mathcal{N}_{\lfloor \bar{p}}$ is then isomorphic to its unfolding, since only isomorphic strings can lead to a merge point, and there are none because of the parsimony condition. This is not sufficient however to make the global trellis $\mathcal{U}^t(\mathcal{N})$ isomorphic to the standard unfolding $\mathcal{U}(\mathcal{N})$, as it was erroneously said in lemma 4 of [45]. A mechanism that would ensure this nice property remains to be found...

5.2.2 Nested co-reflections

Co-reflection of Occ into $Nets$. At this point, we have three nested categories $Occ \subset Tr \subset Nets$. By restricting $Nets$ to Tr in the co-reflection of Occ into $Nets$, we can derive another adjunction between Occ and Tr (another co-reflection, in fact). Specifically, we still have the inclusion functor $F = \subseteq : Occ \to Tr$ in one direction, and the unfolding functor $G = \mathcal{U} : Tr \to Occ$ in the reverse direction. Applying $\mathcal{U}$ to a trellis net $\mathcal{T}$ performs an unfolding in the "conflict dimension" only, since time is already unfolded, so we rather denote by $G = \mathcal{U}^c$ the restriction of $\mathcal{U}$ to Tr.

In this adjunction, the universal property of "conflict unfoldings" still holds (by definition) :

$$\forall \mathcal{T} \in Tr,\ \forall \mathcal{O} \in Occ,\ \forall \phi : \mathcal{O} \to \mathcal{T},\ \ \exists ! \, \psi : \mathcal{O} \to \mathcal{U}^c_{\mathcal{T}},\ \ \phi = f^c_{\mathcal{T}} \circ \psi \tag{5.7}$$

where $f^c_{\mathcal{T}} : \mathcal{U}^c_{\mathcal{T}} \to \mathcal{T}$ is the morphism that refolds conflicts of $\mathcal{U}^c(\mathcal{T})$. Therefore limit preservation theorems allow to state

$$\mathcal{U}^c(\mathcal{T}_1 \times_T \mathcal{T}_2) \quad = \quad \mathcal{U}^c(\mathcal{T}_1) \times_O \mathcal{U}^c(\mathcal{T}_2) \tag{5.8}$$

and give another definition of the product in Occ

$$\mathcal{O}_1 \times_O \mathcal{O}_2 \ \cong \ \mathcal{U}^c(\mathcal{O}_1) \times_O \mathcal{U}^c(\mathcal{O}_2) \quad = \quad \mathcal{U}^c(\mathcal{O}_1 \times_T \mathcal{O}_2) \tag{5.9}$$

These expressions remain valid of course will pullbacks instead of products.

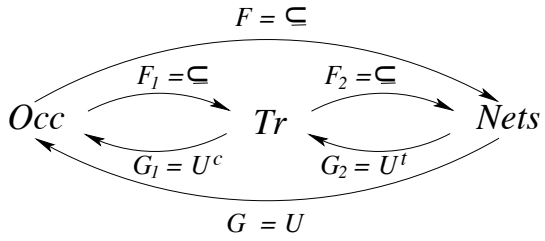


Figure 5.8: *Co-reflections relating categories Occ, Tr and $Nets$.*

Composition of adjunctions. Gathering results obtained so far, we have three adjunctions relating categories Occ, Tr and $Nets$, as displayed by figure 5.8. It is a well known fact that adjunctions can be composed ([81], chap. IV-8, thm 1), so $(F_2 \circ F_1, G_1 \circ G_2)$ defines another pair of adjoints between Occ and $Nets$. But since $F_2 \circ F_1 = F$, we have that $G = G_1 \circ G_2$, up to a natural equivalence[2]. This translates into

$$\forall \mathcal{N} \in Nets, \quad \mathcal{U}(\mathcal{N}) \quad \cong \quad \mathcal{U}^c \circ \mathcal{U}^t(\mathcal{N}) \tag{5.10}$$

and naturally the corresponding foldings can be composed: $f_{\mathcal{N}} = f^t_{\mathcal{N}} \circ f^c_{\mathcal{U}^t_{\mathcal{N}}}$. Equation (5.10) expresses that the time-unfolding $\mathcal{U}^t_{\mathcal{N}}$ of a net can be recovered by "refolding" conflicts on the full unfolding $\mathcal{U}_{\mathcal{N}}$, which we already used when we derived procedure 4 from procedure 2. Specifically, $f^c_{\mathcal{U}^t_{\mathcal{N}}} : \mathcal{U}_{\mathcal{N}} \to \mathcal{U}^t_{\mathcal{N}}$ merges conditions with the same height and representing the same place of $\mathcal{N}$, then merges (or removes) redundant events representing the same transition connected to a given co-set. This can be checked in fig. 5.9 that compares the unfolding and the trellis of our running example.

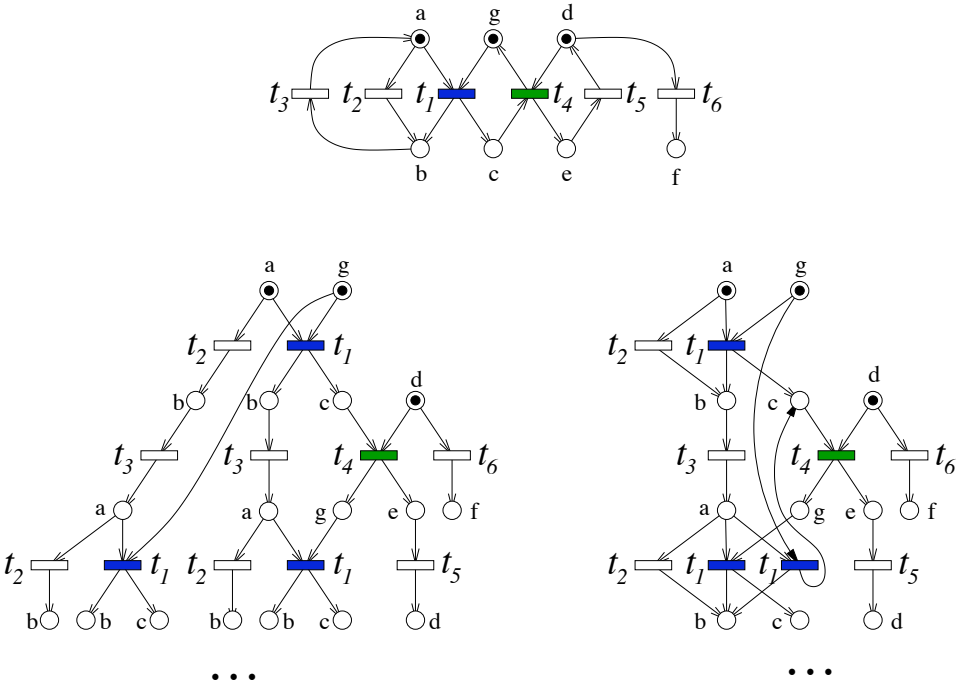


Figure 5.9: *A net $\mathcal{N}$ (top), its unfolding $\mathcal{U}_{\mathcal{N}}$ (bottom left), and its trellis $\mathcal{U}^t_{\mathcal{N}}$ (bottom right).*

In terms of product preservation, the composition of adjoints yields, for any pair

[2]The adjoint of a functor is unique up to a natural equivalence, see [81], chap. IV-1, cor. 1.

$\mathcal{N}_1, \mathcal{N}_2$ in *Nets*

$$\begin{aligned} \mathcal{U}(\mathcal{N}_1 \times \mathcal{N}_2) &= \mathcal{U}(\mathcal{N}_1) \times_O \mathcal{U}(\mathcal{N}_2) && (5.11)\\ &\cong \mathcal{U}^c \circ \mathcal{U}^t(\mathcal{N}_1 \times \mathcal{N}_2) && (5.12)\\ &= \mathcal{U}^c\,[\,\mathcal{U}^t(\mathcal{N}_1) \times_T \mathcal{U}^t(\mathcal{N}_2)\,] && (5.13)\\ &= \mathcal{U}^c\,[\mathcal{U}^t(\mathcal{N}_1)] \times_O \mathcal{U}^c\,[\mathcal{U}^t(\mathcal{N}_2)] && (5.14) \end{aligned}$$

and similarly with pullbacks.

In summary, the category of trellis nets appears as an adjustable intermediate between occurrence nets and safe nets. Its "position" can be adjusted by different choices of height functions, that impose either numerous or scarce merge possibilities.

5.3 Distributed diagnosis : an example

As soon as one has a product operation on trellis nets, which is now granted, a natural notion of projection comes for free. Moreover, thanks to the relation between the unfolding and the trellis of a net, one can recycle the theory developed for distributed computations based on branching processes. Therefore, as long as one doesn't have misleading projections (of trellis processes), the separation theorem holds and allows us to perform distributed computations. The advantage is of course the compactness of these structures. The price to pay lies probably in the difficulty to identify co-sets in a trellis net, a key step in the recursion that computes products.

The algebra that supports distributed computations has been discussed several times, so we rather illustrate the mechanism of trellis based computations on an example. Notice that we don't have (yet) a notion of augmented trellis process, that would allow us to cope with misleading projections, so the example assumes interfaces that reduce to a single site.

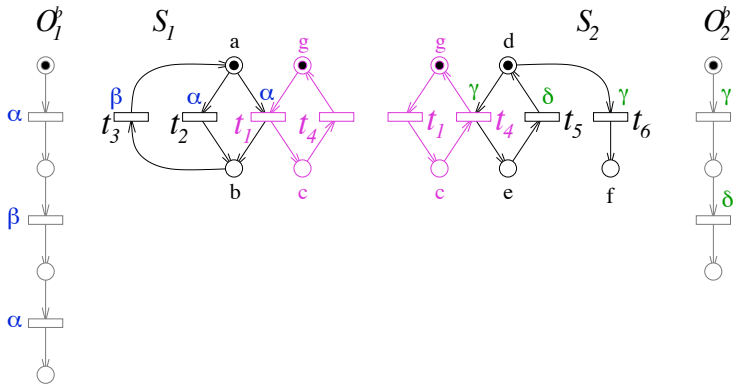


Figure 5.10: *A system $\mathcal{S} = \mathcal{S}_1 \wedge \mathcal{S}_2 = \mathcal{N}_1 \times \mathcal{N}_2 \times \mathcal{N}_3$ with two components, each of them covering two sites : $\mathcal{S}_1 = \mathcal{N}_1 \times \mathcal{N}_3$ and $\mathcal{S}_2 = \mathcal{N}_3 \times \mathcal{N}_2$. The sequences $\mathcal{O}_1^b, \mathcal{O}_2^b$ represent observations on $\mathcal{S}_1, \mathcal{S}_2$ respectively.*

We consider the same setting as in section 4.4.4, with different sequences of observations (Fig. 5.10). The first step consists in computing local diagnoses $\mathcal{D}_i =$

$\mathcal{U}^t_{\mathcal{S}_i} \times_T \mathcal{O}^b_i$, as illustrated in Fig. 5.11. Observe that the conditions of the observed sequences $\mathcal{O}^b_i$ are not duplicated by the product, which contrasts with Fig. 4.18. The events in light gray correspond to discarded parts of $\mathcal{U}^t_{\mathcal{S}_i}$. In the message, the stars indicate possible stop points for $\mathcal{D}_1$. Integrating the message into $\mathcal{D}_2$ doesn't change this local diagnosis: $\mathcal{D}'_2 = \mathcal{D}_2 \wedge_T \Pi_{\mathcal{N}_3}(\mathcal{D}_1) = \mathcal{D}_2$.

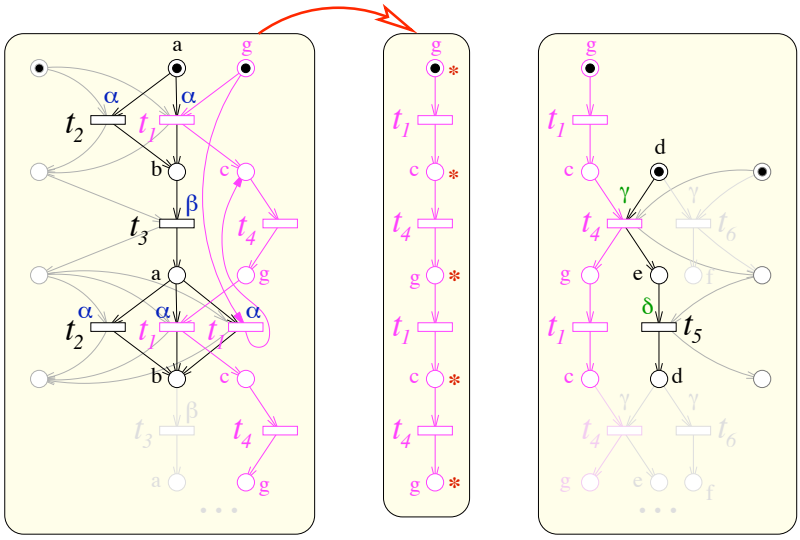


Figure 5.11: *Local diagnoses* $\mathcal{D}_i = \mathcal{U}^t_{\mathcal{S}_i} \times_T \mathcal{O}^b_i$, *and the message* $\Pi_{\mathcal{N}_3}(\mathcal{D}_1)$ *from* $\mathcal{S}_1$ *to* $\mathcal{S}_2$.

The circulation of message $\Pi_{\mathcal{N}_3}(\mathcal{D}_2)$ in the reverse direction essentially prevents $\mathcal{S}_1$ from firing twice transition t_4 (Fig. 5.12), which results in the elimination of a single event of $\mathcal{D}_1$ (in gray). A less visible phenomenon concerns possible stop points in $\mathcal{D}'_1$ (not represented in the figure). The message imposes to produce at least the subsequence (t_1, t_4), which kills in $\mathcal{D}_1$ the purely local explanation (t_3, t_2, t_3).

Taking the combination $\mathcal{D}'_1 \wedge_T \mathcal{D}'_2$ yields the global diagnosis $\mathcal{D}'$. It contains three maximal configurations, or explanations, depicted in Fig. 5.13.

5.4 Summary

For the sequential semantics, we had proposed three structures to encode sub-languages of a system: sets of sequences, branching processes and trellis processes. With a decreasing degree of expressiveness, compensated by an increasing degree of compactness. Algebraically, three properties trigger the possibility of distributed computations: a factorization property on these structures, the existence of projections (consequence of the preceding one), and a separation theorem.

Surprisingly, this formalism can be almost entirely shifted to the partial order semantics, up to some specificities of this framework. Among the aspects that are preserved, the notion of height function, that defines merge points in a trellis process, must still be localized to each elementary component (or site), otherwise the universal property of these objects is lost, which entails the loss of their algebraic

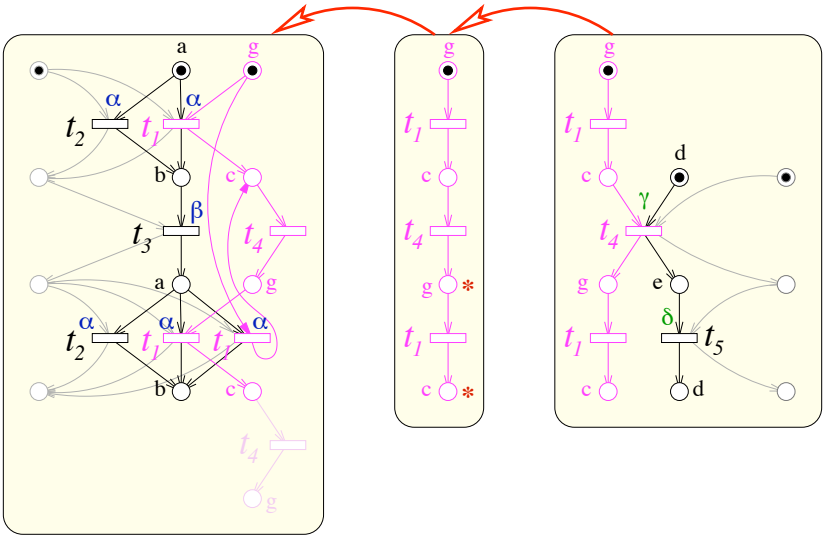


Figure 5.12: *Message* $\Pi_{\mathcal{N}_3}(\mathcal{D}_2)$ *from* $\mathcal{D}_2$ *and its integration into* $\mathcal{D}_1$ *by* $\mathcal{D}_1' = \mathcal{D}_1 \wedge_T \Pi_{\mathcal{N}_3}(\mathcal{D}_2)$.

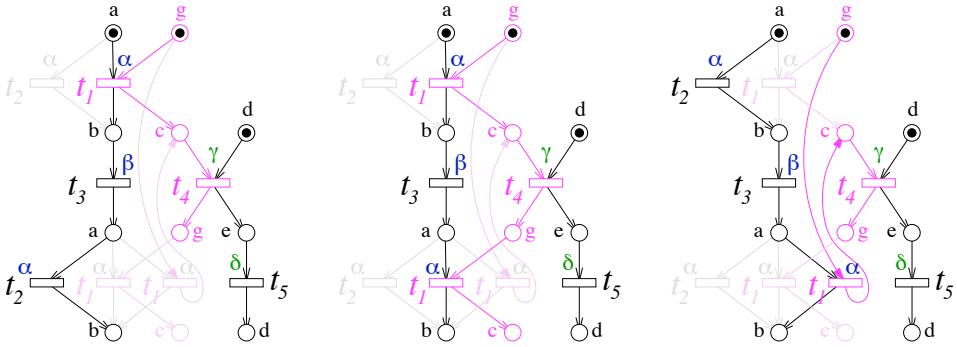


Figure 5.13: *The 3 global solutions to the diagnosis problem.*

properties. By contrast, the simple notion of projection that comes with the product is not sufficient in the true concurrency semantics. In general, it erases useful conflict or causality information, and may result in erroneous computations, excepted in the limited case where components interact by non concurrent interfaces. This limitation motivated the introduction of augmented branching processes, in the previous chapter. Unfortunately, we don't have yet the equivalent notion for trellis processes.

Related work

Surprisingly, almost at the same time, Khomenko *et al.* have proposed a structure very similar to trellis processes, under the name of *merged processes* [68]. These authors develop unfolding-based model checking tools, and were motivated by a memory consumption issue, whence the idea to merge isomorphic futures of configurations. The definition they proposed is slightly different from trellis processes (see [45] for a comparison). In particular, it counts heights separately for each place of $\mathcal{N}$ (instead of sites), and may contain executable circuits. The difficulty to identify co-sets and configurations has been identified, and the authors proposed a solution based on SAT-solvers. Overall, on traditional benchmarks, experiments indicate that one doesn't save much in terms of computation time, but does save in terms of memory space, which is crucial in model checking applications. Merged processes are not universal objects, and so do not enjoy factorization properties. A hot topic is thus to investigate the interest of modular processings for trellis-based model checking. They could suggest more efficient strategies to identify configurations.

Among related works, let us also mention planning problems, and in particular the Graphplan approach [12]. Planing problems consist in organizing elementary tasks in order to reach an objective, so they correspond to reachability problems in model checking. A sub-family of problems, called STRIPS-like domains, considers situations where elementary actions consume and produce local resources, like a Petri net. This community has soon discovered the superiority of partial order methods on state space explorations. The Graphplan approach goes further and represents all possible executions in a domain as a trellis net (see Fig. 2 in [12]). The problem then boils down to optimally exploring this graph, and a planing solution corresponds to a configuration containing the goal. The relevance of partial order semantics is obvious in this context, and there is potentially space for distributed/modular planing algorithms.

Chapter 6

Applications, contracts, technology transfer

The research results presented in this document originated and were stimulated by research contracts, gathering industrial partners as well as other academic teams. The origins date back to CTI contracts (Informal Thematic Collaborations) with France Telecom R&D, around 96, 97, that grew up to take the form of 3 consecutive RNRT[1] projects : MAGDA, MAGDA 2 and SWAN. In the last years, a direct collaboration was initiated with Alcatel to build more realistic prototypes of our alarm correlation algorithms (VDT contract). The decision about their integration into Alcatel's management platform is pending.

This chapter gives an overview of these contracts, for what concerns their distributed diagnosis aspects.

6.1 MAGDA

MAGDA stands for *Modeling and Learning for a Distributed Management of Alarms* (in French). This project (Nov. 98, Nov. 01) was headed by Christophe Dousson (FTR&D). Its main objective was to develop alarm correlation methods, both on-line and off-line, to allow the analysis of alarm logs. An important part of the project was dedicated to correlation methods based on chronicles [5, 28, 29, 78], and to learning methods for chronicles [11, 30, 49, 56].

Self-modeling. The distributed diagnosis part focused on alarm correlation in SDH/SONet networks, as the one depicted in Fig. 6.1. Since we use a model-based approach, a crucial step was the derivation of this model. This task turned out to be the most demanding in terms of research effort. The project jointly came up with a methodology to derive the model, that we now call *self-modeling*. The principle is based on the following ideas :

- There may exist many components in a network, in reality there is a small number of different *types* of components. Here, by component, we refer both to

[1] National Research Network in Telecommunications, funded by the French Ministries of Research and of Industry.

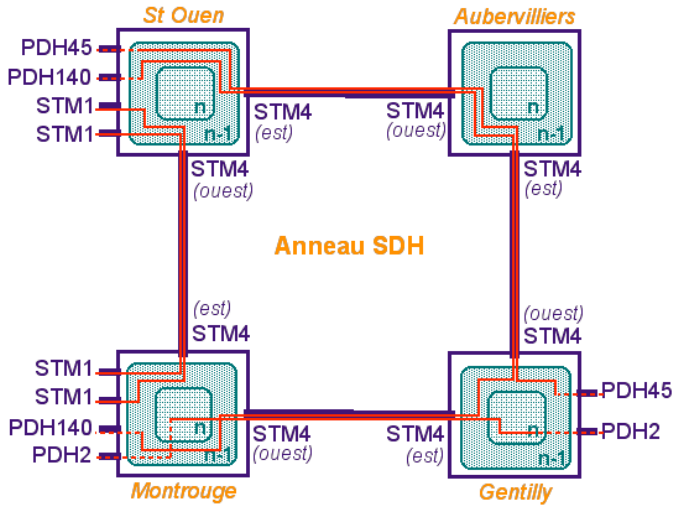


Figure 6.1: *A toy SDH ring, with different connections and displaying the various SDH transmission levels.*

physical equipment, transmission functions in the different layers, adaptation functions, software, etc. The monitoring model, at least its topological part, can be obtained by scanning the network to discover components and their connections, and by instantiating the corresponding "internal model" that will be used for monitoring. The manual task of the modeler thus simply consists in designing the building blocks, *i.e.* generic components, and specifying how their connection capabilities.

- The network behaviors, corresponding to transitions in our monitoring model, can be defined at the scale of components. Elementary behaviors are described partly in the SDH standards, but the most useful information was obtained under the form of failure scenarios, given by an expert. The latter were then decomposed into elementary transitions, described as UML sequence diagrams. Only failure propagations and there consequences were modeled, not the repair or reset actions.

Apparently simple, discovering this methodology, and tuning the model to obtain the expected behaviors was an extremely demanding task. The experimental system contained around 100 elementary components (with 2 to 4 variables per component), encoded into safe Petri nets communicating by shared places.

Diagnosis technology. This first prototype of distributed diagnosis algorithm assumed one local supervisor per network element (so four subsystems, as in Fig. 6.1).

Each supervisor was handling configuration sets, and not branching processes. However, the data structures encoding these sets were in fact branching processes. Once a BP is given, configurations are simply identified by their cut (*i.e.* their maximal conditions). So we were actually handling sets of cuts. A drawback of this approach is that concurrent extensions of configurations generally create trajectories

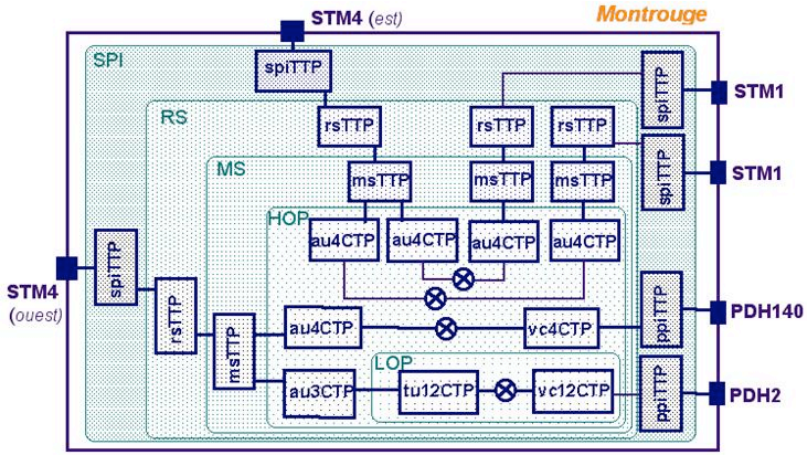


Figure 6.2: *A zoom on the inner components of the network element Montrouge. The figure displays the paths of connections between their injection in the ADM, as high-rate or low-rate connections (HOP/LOP=High/Low Order Path), down to the modulation (SPI=Synchronous Physical Interface). They cross the Multiplex Section (MS) that aggregate connections, and the Regeneration Section (RS). All these components are self-managed and have the ability to raise, transmit or react to alarms.*

that are clones one of another (*i.e.* the underlying branching process violates the parsimony criterion). Therefore these clones had to be detected and removed. By contrast handling configurations has the advantage to facilitate optimization strategies. Each cut was provided with a cost, counting the number of spontaneous failures in the corresponding configuration. It was then easy to perform a local optimization (actually a Viterbi algorithm) on runs examined by each local supervisor : for two runs terminating at the same marking and having the same behavior on the interfaces with neighboring subsystems, only the best one was kept. Finally, a special treatment was introduced for silent transitions, in order to process piles of silent transitions as *macro-tiles*.

In terms of communications, each extension of a run, in a local supervisor, raised an update message when the newly connected transition had an influence on the interface with a neighboring subsystem. Symmetrically, a message received from a neighboring supervisor was used to update local runs, exactly like a local extension.

We did the exercise to fully distribute the algorithm, *i.e.* not to assume any shared memory. Therefore the detection of termination was itself distributed, based on Misra algorithm [90], in order to check that each local supervisor had finished extending its trajectories, and that at the same time no more update message was pending in a communication channel between local supervisors. However, since finite sequences of observations were assumed for each supervisor, the issue of determining a synchronization point where compatible results could be collected didn't appear.

When termination had been detected, a second phase of global optimization was started. This phase exactly implemented a turbo procedure, to determine the most likely global run (recall that the cost of trajectories was optimized locally, for each

local supervisor, not globally).

This algorithm was implemented in JAVA with RMI for communication between local supervisors.

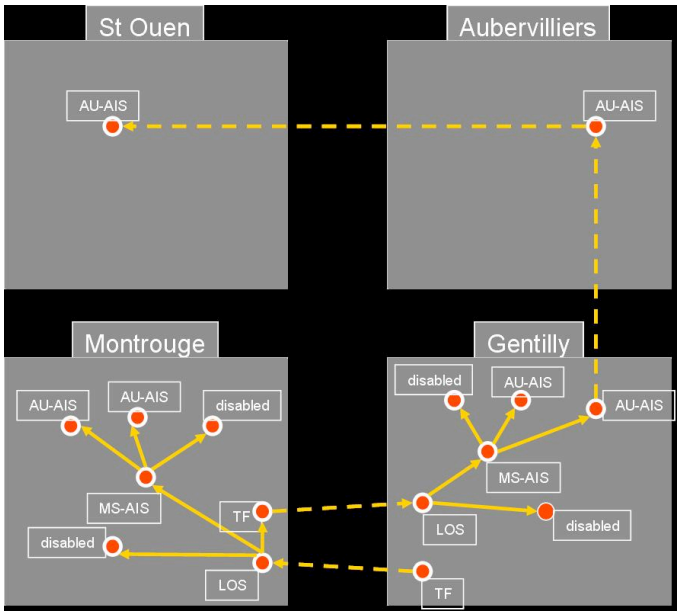


Figure 6.3: *A causal graph of alarms. The root cause in Gentilly (TF=Transmit Failure) corresponds to a laser breakdown.*

Outcomes. As a natural output, we proposed the initial failures in the most likely runs of the system, classified by decreasing likelihood. But the algorithm provided more : the most likely runs actually revealed possible causal links between alarms, which was of major interest to the users. We displayed them under the form of causal graphs of alarms, as the one in Fig. 6.3. The modeling effort was quite rewarding : it allowed us to recognize long-range, topology-dependent correlation patterns, but also to disentangle bursts of alarms generated by multiple failures.

6.2 MAGDA 2

The objectives of MAGDA 2 were more ambitious :

- extending this approach to heterogeneous networks,
- automatically instantiating the model and deploying the distributed supervisors,
- integrating this technology in a standard management platform.

Self-modeling. In terms of self-modeling, the project was very successful. It was shown that the components to supervise belonged to specific classes of managed

objects. The latter can be found in the information model of the chosen network technology, and these information models are quite structured, Roughly speaking, they inherit their elements from common high level definitions of managed objects, like Trail/Connection Termination Point (TTP/CTP), Adaptation Layer, etc. So a great part of the model structure is known : the modeler just has to specialize these classes to the target technology. This is true both for the managed components, for their connection capabilities, and for their behaviors : some behaviors are standard and can be described at a high level, like the transmission of messages between components, state changes in case of failure of a necessary function, the notion of dependence of a service on another, etc.

A tool (OSCAR) was developed to allow a UML drawing of the model. This tool allowed us to draw a topology (deployment diagram), to read one in a file, and was ready to scan a network (which wasn't experimented, however). It also allowed us to refine the managed objects, their connection capabilities, and to design elementary behaviors (so-called *tiles*), with sequence diagrams. Finally, one could define with OSCAR the sub-systems associated to the different local supervisors.

Our application case in this project was a GMPLS/WDM network.

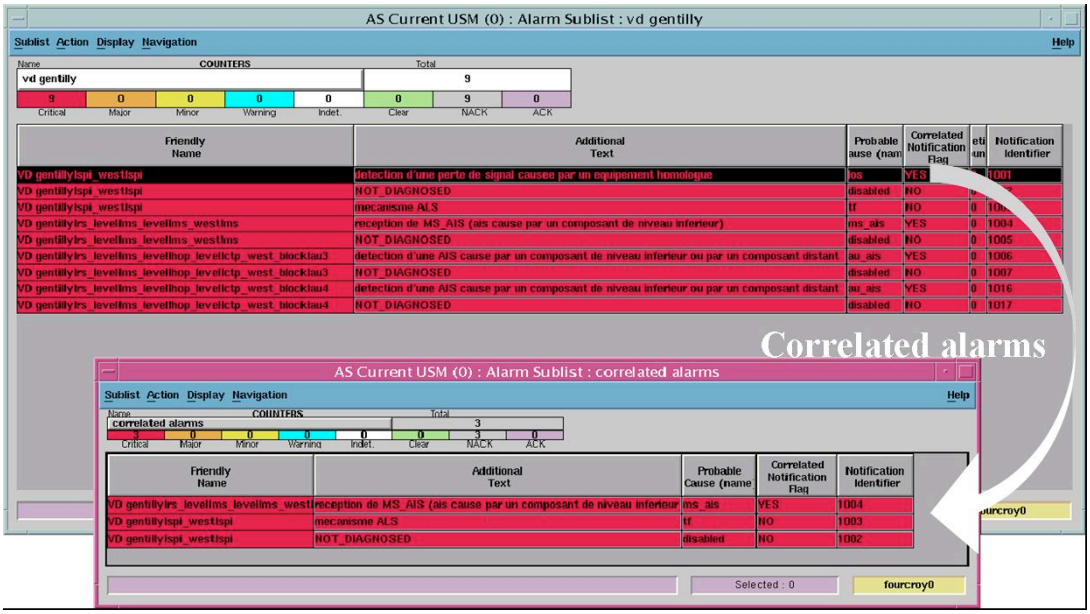


Figure 6.4: *Correlated alarms, as they appeared in the management platform. Root causes were presented first, and the field "correlated notification flag" could be recursively clicked to discover immediate successors of an alarm.*

Diagnosis technology. The computations were still based on configuration sets. The effort concentrated on the following features :

- The automatic deployment of the local supervisors. The latter were sent on different machines, with knowledge of their sub-system model and of their

neighbors, connected together and initialized. This was programmed in Java + CORBA.

- The distributed diagnosis algorithm was implemented above a rule engine, which is the standard correlation technology used in management platforms. We came up with three sets of rules: local extension rules, directly derived from the sub-system model, optimization rules (specific to the management of trajectories), and finally communication rules between local supervisors.

- The direct connection of local supervisors to a true management platform. Both to receive alarms from the platform, and to visualize their correlation at the end of computations. This was done with CORBA.

Outcomes. Programming the distributed diagnosis algorithm over a rule engine was a real challenge, but we proved its feasibility.

In terms of visualization of the results, we only selected the most likely explanation. We translated it into a causality graph between the received alarms, which was used to fill the "correlated" field of alarms as they are stored in the management platform. This is illustrated in Fig. 6.4.

6.3 VDT

Figure 6.5: *A submarine network, with several terminal equipment.*

Objectives. VDT stands for *Viterbi Diagnoser Technology*. The objective of this contract was to propose an alarm correlation method for terminal equipment of submarine lines (SLTE, Fig. 6.5). These network elements aggregate in WDM trans-

missions high rate connections. They involve several layers of multiplexing and amplification, which results in quite complex systems, that can easily fill several racks (Fig. 6.6). The objective was to correlate failure information propagating between these functions. For example the fact that a low output signal of an amplifier will cause extraction errors on all wavelengths multiplexed in this signal.

The major difficulty was that the propagation of failures were easy to describe at the physical level. Normally, this is where the redundancy of alarms should be avoided, by adequate masking procedures involving simple communications between cards. Unfortunately, this feature is often considered as secondary, and delivery delays generally prevent its development. So the correlation must be performed at the management level. There is no one-to-one mapping between the managed objects and the underlying functions that support them. In general, a managed object, like an Optical Path or an Optical Channel Group, collects information from several cards, and symmetrically a failure on a given card is reflected in several managed objects. Moreover, although alarms are non ambiguous at the physical level, they are translated (when they are) into very generic failure indications at the management level.

Specifying the system model at the management level was a real challenge. Nevertheless, we managed to recycle an important part of the self-modeling methodology developed in Magda and Magda 2. A few components, specific to this technology, had to be designed, as well as specific connection capabilities. Behaviors were again extracted from failure scenarios given by an expert. And the counterpart of the OSCAR tool was redeveloped to easily draw the model.

Figure 6.6: *Two submarine line terminal equipment (SLTE).*

Diagnosis technology. This time we experimented a true unfolding-based approach, in a centralized setting (a single supervisor). It was fed on one side by the model produced by OSCAR, which was itself built from the description of an equipment architecture. On the other side, the diagnosis algorithm was fed by alarm logs, taking the form of a partial order (more precisely, a tuple of sequences). We experimented different unfolding algorithms. The most efficient was based on a recursive sub-routine to discover co-sets where a possible extension could be performed. The unfolding was guided both by observations and by a cost function, in order to favor

configurations minimizing the number of root failures. As output, the algorithm provided both the unfolding (Fig. 6.7) and a display of the SLTE topology identifying the managed objects responsible for the root failures.

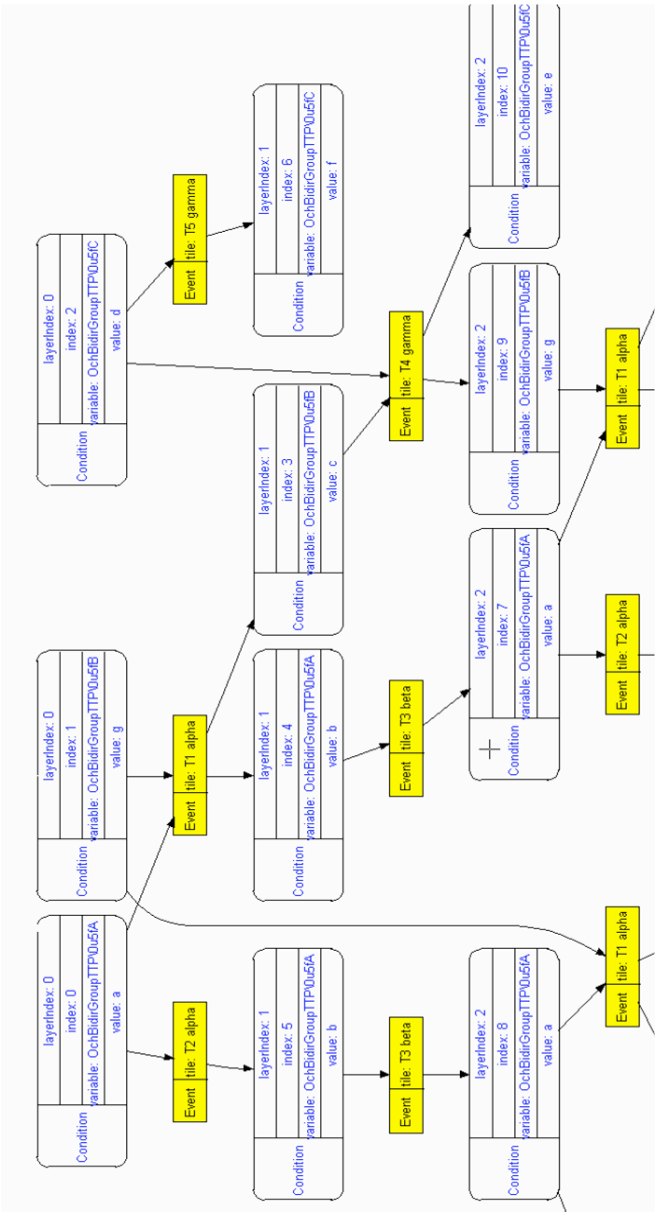


Figure 6.7: *Part of the unfolding representing possible failure propagations.*

Chapter 7

Conclusion

7.1 Summary of results

In substance, this document proposes a methodology to represent the trajectory set of a complex system, and an algebra to perform computations on these trajectories. We started by the definition of what we call complex systems : they are obtained by assembling elementary components into a network, and thus can be described as a graph of components. The time dimension is introduced by "unfolding" the system, in different ways, which is performed by a *functor* applied to the system. Many efforts have been dedicated to the design of such functors, in order to obtain transformations that would preserve the interaction structure between components. So in all the proposed settings, runs of a compound system can be expressed as the combination of runs of its components, which is called the "factorization property." Formally, this property is equivalent to the factorization of the probability distribution in a Markov random fields, which is itself the translation of conditional independence statements between components of the field (see the Hammersley-Clifford theorem). The analogy of these two domains has been established by means of a common axiomatic framework. As a consequence, optimal estimation algorithms developed for Markov random fields (or Bayesian networks) can be recycled into distributed asynchronous algorithms for networks of dynamic systems. More specifically, beyond factorization aspects, this formal analogy requires the existence of a notion of projection that allows us to express a form of conditional independence statement on trajectory sets (axiom (a3)).

The methodology presented in this document is very flexible and adapts to different choices of trajectory semantics, and trajectory set representations. The two arrays below summarize results.

In the case of sequential semantics, one can represent trajectory sets as a language (*i.e.* a collection of sequences), as a branching process (*i.e.* a decision tree), or as a trellis process, which extends the usual notion of trellis of an automaton. In all cases, these structures factorize into simpler factors, and there exists a natural notion of projection that allows us to implement message passing algorithms, for example to solve the diagnosis problem. When runs are provided with costs, it is sometimes possible to use the MPA to perform an optimization task, for example to determine

	Sequential semantics		
	language systems	branching processes	trellis processes
factorization property	yes	yes	yes
existence of projections	yes	yes	yes
optimization techniques	yes	yes	?

Figure 7.1: Available results for sequential semantics.

the global run of the system that minimizes the cost. We didn't present it for trellis processes, whence the question mark, but this result seems accessible. It would take the form of cooperating Viterbi algorithms, one per component, where several best cost values (instead of one) would be carried by each state. They would correspond to the different synchronization patterns with neighboring components.

However, sequential semantics seem inappropriate for distributed systems : they generally induce large structures, whatever the representation one chooses. This is due to two phenomena : first of all, the multi-clock feature, necessary to the factorization property, imposes large trellises, and secondly the natural concurrency that arises in modular systems is not handled appropriately, which uselessly multiplies the number of runs to consider. So the interest of sequential semantics is essentially theoretical : it provides a simple setting where the structure of distributed computations can be explained quite easily.

	True concurrency semantics		
	Mazurkiewicz traces (configurations)	branching processes	trellis processes
factorization property	yes	yes	yes
projections on a single variable	yes	yes	yes
existence of general projections	yes	yes	?
optimization techniques	yes	?	?

Figure 7.2: Available results for true concurrency semantics.

We rather recommend to use true concurrency semantics, that considerably reduce the number of trajectories to consider, since the interleavings of concurrent events are not computed. The three representations above remain possible. Instead of languages, one has sets of configurations, or equivalently sets of Mazurkiewicz traces. Branching processes are now less trivial than simple decision trees : they become prefixes of the system unfolding, as it was defined for Petri nets. And finally, we have also defined the appropriate notion of trellis process, an extension of the notion of trellis to concurrent systems. Projections exist in almost all cases : they are easy to define when one projects on a simple system, with a single variable (and thus no possibility of internal concurrency). In the general case, one must keep track of some conflict and causality relations, which requires the more complex notion of augmented configuration or augmented branching process. This general projection is still missing for trellis processes. Optimization techniques can also be

implemented on sets of traces : they look pretty much like those on languages. But at this point it is not known whether they extend to the other structures.

7.2 Directions for future work

7.2.1 Technical extensions

The framework presented in this document may look quite complete and stable, but in reality many technical extensions demanded by applications remain open questions. The most obvious ones are the empty entries in the previous arrays. In particular, a clear understanding of optimization issues is still missing. But several other research directions deserve attention :

Label structures. The formal framework of chapter 2 is based on systems interacting by shared variables. But the computations we have presented, based on compact encodings of trajectory sets, do not fit exactly this setting : interactions are better described by shared labels, and projections on label sets are not (yet) defined. This motivated the use of pullbacks, to express component interactions by means of shared variables/sites, *i.e.* interface automata. These representations remain a bit unnatural because, in a pullback, the shared sites do not necessarily capture all interactions between two components. Two ways to clarify this setting : either by re-expressing results of chapter 2 with a new formalism based on label interactions (in the spirit of the separation criterion appearing in theorem 9), or by developing labeled event structures and a way to compose them with trajectory sets (as in section 3.4.2).

On-line computations. A nice algebraic translation of recursive algorithms in the category theory setting is missing : what does it mean to replace an observed sequence by a longer one. Understanding this has important practical consequences : When one runs a recursive distributed diagnosis algorithm, messages are exchanged and new observations are introduced in a chaotic manner. One would like to determine on-line which part of the local diagnoses computed so far are "stable" and can be displayed. In the same way, where and how can we set a temporary stop point in the algorithm to observe its current result. This certainly relates to the scheduling of operations in the algorithm, that was an unused degree of freedom so far. It also relates to "pure" aspects of distributed computing, as studied in [90].

Symbolic unfoldings. This idea was introduced by Thomas Chatain and Claude Jard. Consider a "generic tile" t that would read the value of a state variable V and add one to it. In the traditional construction of unfoldings, all values of V would appear as (conflicting) conditions, and the action of t would be connected to each of them. In symbolic unfoldings, variable V appears only once, with an unknown value that depends on its past, and the action of t is represented as a symbolic event adding 1 to V. This results in much more compact structures. The price to pay is a more expensive test to decide whether a tile can be connected or not. In order to compute with these objects, Thomas and I have already proved

that the factorization property still holds. It remains to build the adequate notion of projection. Let us mention that symbolic unfoldings are a natural tool to unfold time systems.

Multiresolution/hierarchical algorithms. The advantage of systems defined by constraints is that redundant constraints can only help reduction algorithms. The point is to understand where ! One can suspect that extra long range constraints could help to reach a better convergence of message passing algorithms (MPA). One could even imagine that well designed redundant components could help separate the interaction graph into smaller parts. This is of course a model transformation issue, and there are probably resources about this question in the literature about data-bases. Notice in particular that MPA are known in that field under the name of query/sub-query algorithms.

Robustness issues. For our applications related to telecom networks, we have developed a methodology to automatically build a model of the supervised network, by assembling generic components. In practice, a complete model is not always available. One may only know parts of it, for example when a single domain is monitored in a multi-domain architecture, or when a specific intermediate network layer is monitored. Moreover, modeling errors may also introduce a mismatch between what a model could produce, and what the system actually outputs. Finally, in the case of truly distributed algorithms, messages exchanged by local supervisors could be lost, for example when the network management information is transmitted in-band. All these robustness issues have not been explored so far.

7.2.2 Research directions

Being able to compute efficiently on the trajectory set of a complex system potentially opens the way to numerous applications. We had distributed diagnosis issues as our target, but several other domains seem accessible, or are worth being explored. Let us mention :

Smooth systems. This name suggests systems that have a non rigid interaction structure. Networks can be reconfigured for example, create or kill connections, which amounts to creating or removing components in their model. In web services, the tree (or graph) of service requests, also called its choreography, may depend on the parameters of the initial request or on the answer of some sub-services. The same phenomena appear with active XML documents (see the ASAX project). The difficulty here is not so much to define a notion of unfolding for such systems, where some transitions may create or disconnect components, but rather to propose a convenient model to describe the structural evolutions. Some partial answers already exist, and there is hope to obtain factorization properties. But we are still far from applications.

Ad-hoc modeling. Related to modeling issues, another research direction concerns the extension of this work to other formalisms of concurrent systems. Scenario

languages, or various forms of temporal logics could be explored. Defining the adequate level of description of a system is also an interesting topic : in general, one is not so much interested in computing all runs of a system that explain observations, but rather in checking a higher level property like "did this phenomenon occur in the runs that produced this observation ?" In general, a system model contains more details than necessary to check a given set of such properties. There is no obvious way so far to automatically determine what minimal granularity should be preserved in a model in order to check these properties with the same accuracy.

Modular model checking. An important application area of unfolding techniques concerns model checking problems, for example in circuit design. Questions like deadlock freeness, liveness, accessibility of a state or of a transition are examined. A central point to address these problems is the construction of a finite complete prefix of the unfolding, in order to capture all behaviors of interest for a given system and a given property to check. So far, this notion doesn't exist for modular systems. In cooperation with Victor Khomenko (Newcastle upon Tyne, UK) and Agnes Madalinski (ex Newcastle student, now postdoc in Rennes), we are currently exploring the construction of prefixes in product form, that would of course be more compact and allow verifications by parts.

Distributed optimal control. This is a more futuristic (optimistic ?) objective, based on the following intuition : The Bellman equation solves the optimal control problem of a Markov chain, given past observations on that chain. A very similar recursion appears in the Viterbi algorithm, used to recover the most likely trajectory of a chain given observations. Both solutions are actually based on the dynamic programming principle. Pushing further this remark, one could imagine that a solution to the distributed optimal trajectory reconstruction (one of the missing entries in the above arrays) would suggest strategies to design distributed optimal control algorithms for modular systems.

Optimal planning. The problem consists in organizing a huge set of possible tasks in order to achieve a given objective. Formally, this is an accessibility problem, for a target state (called the goal), in a system containing large sets of variables and tiles. Concurrency is natural in this setting. Traditional approaches, developed in the AI community, use unfolding-like structures to encode sets of runs. In particular, in the Graphplan approach, the proposed structure is very close to what we called the time-unfolding. The possibility of modular computations has not been explored in this field. And modular optimization would of course be a breakthrough. A collaboration project is in preparation with Sylvie Thiebaut (Canberra).

Information theory for distributed systems. Here we are beyond futuristic dreams... In several places of this document, we have underlined the relations between Bayesian networks and distributed dynamic systems. We would like to push further this analogy, provided networks of dynamic systems can be properly randomized, in order to introduce information theory in this setting. For example to

quantify the effect of a component on the others, evaluate the importance of a message in distributed algorithms, etc. This is probably a difficult task, given the poor number of results available in network information theory, or in network coding.

Chapter 8

Acknowledgement

The School of Athens in Distributed Diagnosis

This picture was taken on June 14th, 2007, on the way out of IRISA's conference room, where the defense was taking place. Yes, we do have such magnificient rooms at IRISA, but we generally adopt a different dress code. Excepted on special occasions. That day was one of these occasions, as we were visited by many people of wisdom.

I am very indebted to Albert Benveniste, whom we see here, pointing his hand down to applications. I feel unable to explain how much he supported this work. As a discoverer of fruitful industrial problems, as a supporter of new approaches to a problem, as a scientific reference, as a clarifier of confuse ideas, etc. My gratitute won't find adequate words.

This is Glynn Winskel, pointing his finger up to the ideal world of theoretical approaches, and carrying a copy of his *Lecture Notes in Category Theory*. I already mentioned how much his contributions have been inspiring for my work. I feel scientifically indebted to him, and honored that he accepted to give his opinion on such non-standard uses of formal methods.

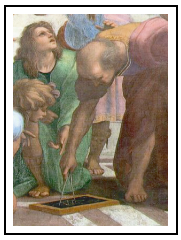
Alan Willsky, saying it's highly time to go to the restaurant. Alan has the talent to conjugate a powerful scientific inspiration, with a deep and educated taste for the good aspects of life. Visiting him at MIT has always been stimulating on both topics ! His scientific enthousiasm is always a refreshing pleasure, and I'm glad he accepted to evaluate my deviating uses of belief propagation algorithms.

Nothing is as stimulating as a strong competitor. Stéphane Lafortune has laid a corner stone in diagnosis problems and diagnosability issues, including distributed aspects. After several years of proximity in conference sessions, where you wait for his last novelties to nurture your own reflections, it's a pleasure and a challenge to know his evaluation of a different point of view on the topic.

There are some people you're always pleased to meet in a conference. Alessandro Giua is one of them, both for his pleasant company and for his wide culture in computer science, Petri nets, discrete event systems, control... A challenge to have him as a reviewer.

Christophe Dousson has been our light house on industrial problems for several years. He is himself a contributor to diagnosis solutions, and shared with us his accurate perception about emerging technologies and the problems they raised. Industrial partnerships have been crucial to my work.

This is Michel Raynal, teaching. His books about distributed computing helped me understand how to make the joint between modular processings and truly distributed algorithms. Among his (numerous) motto *Correctness may be theoretical, but incorrectness has a practical impact.* I'm honored he accepted to venture a bit outside his land to evaluate this document.



Your servant, exhausted.
All this work for a parchment...

There are many other persons that directly or indirectly contributed to this work, as colleagues or as friends, with stimulating ideas, friendly support or love. Let me just mention Armen Aghasaryan, and Claire. I'm sure the others will recognize themselves on the picture.

Bibliography

[1] S. Abbes, "Probabilistic Models for Distributed and Concurrent Systems. Limit Theorems and Applications to Statistical Parametric Estimation," PhD thesis no. 3026, IRISA, University of Rennes 1, Oct. 2004.

[2] S. Abbes, A. Benveniste, "Probabilistic models for true-concurrency : branching cells and distributed probabilities for event structures," Information & Computation 204(2), pp. 231-274, Feb. 2006.

[3] S. M. Aji, R. J. McEliece, "The Generalized Distributive Law," IEEE Trans. Inform. Theory, vol. 46, no. 2 (March 2000), pp. 325-343.

[4] S. M. Aji, R. J. McEliece, "The Generalized Distributive Law and Free Energy Minimization," 39th Allerton Conference, October 4, 2001.

[5] A. Aghasaryan, C. Dousson, "Mixing Chronicle and Petri Net Approaches in Evolution Monitoring Problems," 12th workshop on Principles of Diagnosis (DX'01), pp. 1-7, March 2001.

[6] A. Arnold, "Finite Transition Systems," Prentice Hall, 1992.

[7] P. Baldan, A. Corradini, H. Montanari, "Contextual Petri nets, asymmetric event structures, and processes," Information and Computation, Vol. 171(1), pp. 1-49, Nov. 2001.

[8] P. Baroni, G. Lamperti, P. Pogliano, M. Zanella, "Diagnosis of Large Active Systems," Artificial Intell. 110, pp. 135-183, 1999.

[9] A. Benveniste, E. Fabre, S. Haar, C. Jard, "Diagnosis of asynchronous discrete event systems, a net unfolding approach," IEEE Trans. on Automatic Control, vol. 48, no. 5, pp. 714-727, May 2003.

[10] C. Berrou, A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," In IEEE Trans. on Communications 44(10), Oct. 1996.

[11] S. Bibas, M.-O. Cordier, P. Dague, C. Dousson, F. Lévy, L. Rozé, "Alarm driven supervision for telecommunication networks: I- Off-line scenario generation," Annals of Telecommunications n° 9/10 (tome 51),,pp. 493-500, Sept./Oct. 1996.

[12] A. Blum, M. Furst, "Fast Planning Through Planning Graph Analysis," , Artificial Intelligence, 90:281–300, 1997.

[13] R.K. Boel, J.H. van Schuppen, "Decentralized Failure Diagnosis for Discrete Event Systems with Costly Communication between Diagnosers," in Proc. 6th Int. Workshop on Discrete Event Systems, WODES'02, pp. 175-181, 2002.

[14] R.K. Boel, G. Jiroveanu, "Distributed Contextual Diagnosis for very Large Systems," in Proc. of WODES'04, pp. 343-348, 2004.

[15] C. Cassandras, S. Lafortune, "Introduction to Discrete Event Systems," Kluwer Academic Publishers, 1999.

[16] F. Cassez, T. Chatain, C. Jard, "Symbolic unfoldings for networks of timed automata," In ATVA, LNCS 4218, pp. 307-321, 2006.

[17] I. Castellani, G.-Q. Zhang, "Parallel product of event structures," Theoretical Computer Science, no. 179, pp. 203-215, 1997.

[18] T. Chatain, C. Jard, "Complete finite prefixes of symbolic unfoldings of safe time Petri nets," ATPN'06, LNCS 4024, pp. 125-145, 2006.

[19] T. Chatain, "Symbolic Unfoldings of High-Level Petri Nets and Application to Supervision of Distributed Systems," PhD thesis, Université de Rennes 1, November 2006.

[20] S.-Y. Chung, T. Richardson, R. Urbanke, "Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation," IEEE Trans. Inform. Theory, 47 (2001), pp. 657-670.

[21] O. Contant, S. Lafortune, "Diagnosis of Modular Discrete Event Systems," in Proc. of WODES'04, pp. 337-342, 2004.

[22] O. Contant, S. Lafortune, "Diagnosability of Discrete Event Systems with Modular Structure, J. of Discrete Event Dyn. Syst., vol. 16, pp. 9-37, 2006.

[23] J.-M. Couvreur, S. Grivet, D. Poitrenaud, "Unfolding of Products of Symmetrical Petri Nets," 22nd International Conference on Applications and Theory of Petri Nets (ICATPN 2001), Newcastle upon Tyne, UK, June 2001, LNCS 2075, pp. 121-143.

[24] P. Darondeau, "Distributed implementations of Ramadge-Wonham supervisory control with Petri nets," In 44th Conf. on Decision and Control (CDC), Seville, Spain, pp. 2107-2112, December 2005.

[25] R. Debouk, S. Lafortune, D. Teneketzis, "Coordinated Decentralized Protocols for Failure Diagnosis of Discrete Event Systems," Discrete Event Dynamic Systems, vol. 10(1/2), pp. 33-86, 2000.

[26] P. Degano, R. De Nicola, U. Montanari, "On the Consistency of Truly Concurrent Operational and Denotational Semantics," in proc. Symposium on Logic in Computer Science (LICS) 1988, pp. 133-141.

[27] V. Diekert, G. Rozenberg, "The Book of Traces," World Scientific Publishing, 1995.

[28] C. Dousson, "Alarm driven supervision for telecommunication networks: II- On-line chronicle recognition," Annals of Telecommunications n° 9/10 (tome 51), pp. 501-508, Sept./Oct. 1996.

[29] C. Dousson, P. Le Maigat, "Chronicle Recognition Improvement Using Temporal Focusing and Hierarchization," 20th International Conference on Artificial Intelligence (IJCAI'07), pp. 324-329, Jan. 2007.

[30] C. Dousson, T. Vu Duong, "Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems," 16th IJCAI, pp. 620-626, Stockholm, Sweden, Aug. 99.

[31] J. Engelfriet, "Branching Processes of Petri Nets," Acta Informatica no. 28, pp. 575-591, 1991.

[32] J. Esparza, "Model checking using net unfoldings," Science of Computer Programming 23, pp. 151-195, 1994.

[33] J. Esparza, S. Römer, W. Vogler, "An improvement of McMillan's unfolding algorithm," in Proc. of TACAS'96, LNCS 1055, pp. 87-106.

[34] J. Esparza, S. Römer, W. Vogler, "An Improvement of McMillan's Unfolding Algorithm," Formal Methods in System Design 20(3), pp. 285-310, May 2002. Extended version of [33].

[35] J. Esparza, S. Römer, "An unfolding algorithm for synchronous products of transition systems," in Proc. of CONCUR'99, LNCS 1664, Springer Verlag, 1999.

[36] J. Esparza, C. Schröter, "Reachability Analysis Using Net Unfoldings," Workshop of Concurrency, Specification and Programming, volume II of Informatik-Bericht 140, pp. 255-270, Humboldt-Universität zu Berlin, 2000.

[37] E. Fabre, "Factorization of Unfoldings for Distributed Tile Systems, Part 1 : Limited Interaction Case," INRIA research report no. 4829, April 2003.

[38] E. Fabre, "Convergence of the turbo algorithm for systems defined by local constraints," INRIA research report no. PI 4860, May 2003.

[39] A. Benveniste, E. Fabre, S. Haar, "Markov Nets: Probabilistic Models for distributed and concurrent systems," IEEE Transactions on Automatic Control, 48(11):1936-1950, November 2003.

[40] E. Fabre, "Factorization of Unfoldings for Distributed Tile Systems, Part 2 : General Case," INRIA research report no. 5186, May 2004.

[41] E. Fabre, A. Benveniste, S. Haar, C. Jard, "Distributed Monitoring of Concurrent and Asynchronous Systems," Journal of Discrete Event Dynamic Systems, special issue, vol. 15 no. 1, pp. 33-84, March 2005.

[42] E. Fabre, "Distributed Diagnosis based on Trellis Processes," In 44th Conf. on Decision and Control (CDC), Seville, Spain, pp. 6329-6334, December 2005.

[43] E. Fabre, "On the Construction of Pullbacks for Safe Petri Nets," In Applications and Theory of Petri Nets and other Models of Concurrency, ATPN'06, Turku, Finland, LNCS 4024, pp. 166-180, June 2006.

[44] E. Fabre, C. Hadjicostis, "A Trellis Notion for Distributed System Diagnosis with Sequential Semantics," In proc. 8th Int. Workshop on Discrete Events Systems, WODES, Ann Arbor, pp. 294-300, July 2006.

[45] E. Fabre, "Trellis Processes : a Compact Representation for Runs of Concurrent Systems," To appear in Journal of Discrete Event Dynamical Systems, 2007.

[46] E. Fabre, A. Benveniste, "Partial Order Techniques for Distributed Discrete Event Systems : Why You Can't Avoid Using Them." Plenary address presented by A. Benveniste, 8th Int. Workshop on Discrete Events Systems, WODES, Ann Arbor, pp. 1-2, July 2006.

[47] E. Fabre, A. Benveniste, "Partial Order Techniques for Distributed Discrete Event Systems : Why You Can't Avoid Using Them." INRIA Research Report RR 5916, May 2006, submitted to J. Discrete Event Dyn. Syst.

[48] E. Fabre, "Distributed system diagnosis with sequential semantics," in preparation.

[49] F. Fessant, C. Dousson, F. Clérot, "Mining of a telecommunication alarm log to improve the discovery of frequent patterns," Industrial Conf. on Data Mining (ICDM'04), Leipzig, Germany, July 2004.

[50] C.J. Fidge, "Logical time in distributed computing systems." *IEEE Computer* **24**(8), 28–33, 1991.

[51] A. Giua, X. Xie, "Control of safe ordinary Petri nets with marking specifications using unfolding," Proc. IFAC WODES04: 7th Workshop on Discrete Event Systems (Reims, France), Sept. 2004.

[52] A. Giua, X. Xie, "Nonblocking control of Petri nets using unfolding," 16th IFAC World Congress (Prague, Czech Republic), July 2005.

[53] A. Giua, X. Xie, "Control of safe ordinary Petri nets using unfolding," J. of Discrete Event Dynamic Systems, Vol. 15(4), pp. 349-373, Dec. 2005.

[54] S. Genc, S. Lafortune, "Distributed Diagnosis of Discrete-Event Systems Using Petri Nets," in proc. 24th Int. Conf. on Applications and Theory of Petri Nets, LNCS 2679, pp. 316-336, June, 2003.

[55] S. Genc and S. Lafortune, "Distributed Diagnosis of Place-Bordered Petri Nets," accepted for publication to the IEEE Transactions on Automation Science and Engineering, March 31, 2006.

[56] B. Guerraz, C. Dousson, "Chronicles Construction Starting from the Fault Model of the System to Diagnose," Int. Workshop on Principles of Diagnosis (DX'04), pp. 51-56, Carcassonne, France, June 2004.

[57] A. Guyader, E. Fabre, C. Guillemot, M. Robert, "Joint source-channel turbo decoding of entropy-coded sources," IEEE Journal on Selected Areas in Communications, Vol. 19(9), pp. 1680-1696, Sept. 2001.

[58] J Hagenauer, "The EXIT chart - Introduction to extrinsic information transfer in iterative processing," In Proc. 12th Eur. Signal Proc. Conf., EUSIPCO '04, pp. 1541-1548, Vienna, Sept. 2004.

[59] K.X. He, M.D. Lemmon, "Liveness-Enforcing Supervision of Bounded Ordinary Petri Nets using Partial Order Methods," IEEE Trans. on Automatic Control, vol 47, pp. 1042-1055, July 2002.

[60] K. Heljanko, V. Khomenko, M. Koutny, "Parallelisation of the Petri Net Unfolding Algorithm," 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS/ETAPS 2002), Grenoble, France, LNCS 2280 pp. 371-385, 2002.

[61] T. Jeron, H. Marchand, S. Pinchinat, M.-O. Cordier, "Supervision Patterns in Discrete Event Systems Diagnosis," in proc. of WODES'06, pp. 262-268, Ann Arbor, July 10-12, 2006.

[62] G. Jiroveanu, R. Boel, "Distributed Diagnosis for Petri Net models with unobservable interactions via common places," 11th Conf. on Decision and Control (CDC'05, pp. 6305-6310, 2005.

[63] G. Jiroveanu, R. Boel, "Petri Net model-based Distributed Fault Diagnosis for large Interacting Systems." 16th Int. Workshop on Principles of Diagnosis (DX'05), pp. 25-30, 2005.

[64] V. Khomenko, M. Koutny, W. Vogler, "Canonical Prefixes of Petri Net Unfoldings," 14th Int. Conf. on Computer Aided Verification (CAV 2002), Copenhagen, Denmark, LNCS 2404, pp. 582-595, 2002.

[65] V. Khomenko, M. Koutny, "LP Deadlock Checking Using Partial Order Dependencies," 11th Int. Conf. on Concurrency Theory (CONCUR 2000), University Park, Pennsylvania, USA, LNCS 1877, pp. 410-425, 2000.

[66] V. Khomenko, M. Koutny, "Towards an Efficient Algorithm for Unfolding Petri Nets," 12th Int. Conf. on Concurrency Theory (CONCUR 2001), Aalborg, Denmark, LNCS 2154, pp. 366-380, 2001.

[67] V. Khomenko, M. Koutny, W. Vogler, "Canonical Prefixes of Petri Net Unfoldings," Acta Informatica, vol. 40, pp. 95-118, 2003

[68] V. Khomenko, A. Kondratyev, M. Koutny, W. Vogler, "Merged Processes - a New Condensed Representation of Petri Net Behaviour," Tech. Rep. Series CS-TR-884, Univ. of Newcastle upon Tyne, Jan. 2005.

[69] F. R. Kschischang, B. J. Frey, "Iterative Decoding of Compound Codes by Probability Propagation in Graphical Models," IEEE J. on Selected Areas in Communications, vol. 16(2), Feb. 1998, pp. 219-230.

[70] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," IEEE Transactions on Information Theory, vol. 47(2), Feb. 2001, pp. 498-519.

[71] G. Lamperti and M. Zanella. Diagnosis of discrete-event systems from uncertain temporal observations. *Artif. Intell.* 137(1-2): 91-163 (2002).

[72] G. Lamperti and M. Zanella. *Diagnosis of Active Systems: Principles and Techniques.* Kluwer International Series in Engineering and Computer Science, Vol. 741, 2003.

[73] G. Lamperti and M. Zanella. Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. *Artif. Intell.* 170(3): 232-297 (2006).

[74] G. Lamperti and M. Zanella. Incremental Processing of Temporal Observations in Supervision and Diagnosis of Discrete-Event Systems. ICEIS (2) 2006: 47-57.

[75] L. Lamport, N. Lynch, "Distributed Computing : Models and Methods," in Handbook of Theoretical Computer Science, vol. B : Formal Models and Semantics, Jan van Leeuwen ed., Elsevier (1990), pp. 1157-1199.

[76] S. L. Lauritzen, D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," J. Royal Statistical Society, Series B, vol. 50(2), pp. 157-224, 1988.

[77] S. L. Lauritzen, "Graphical Models," Oxford Statistical Science Series 17, Oxford University Press, 1996.

[78] P. Le Maigat, C. Dousson, "Improvement of Chronicle-based Monitoring using Temporal Focalization and Hierarchization," 17th International Workshop on Principles of Diagnosis (DX'06), pp. 257-261, June 2006.

[79] R. J. McEliece, E. R. Rodemich, J.-F. Cheng, " The Turbo Decision Algorithm," 33rd Allerton Conf. on Communication, Control and Computing, Oct. 1995.

[80] R. J. McEliece, D. J. C. MacKay, J.-F. Cheng, "Turbo Decoding as an Instance of Pearl's Belief Propagation Algorithm," IEEE Journal on Selected Areas in Communication, vol. 16(2), Feb. 1998, pp. 140-152.

[81] S. Mac Lane, "Categories for the Working Mathematician," Springer-Verlag, 1971.

[82] K.L. McMillan, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits," in Proc. 4th Workshop of Computer Aided Verification, Montreal, 1992, pp. 164-174.

[83] K.L. McMillan, "Symbolic Model Checking: An Approach to the State Explosion Problem," PhD thesis, Kluwer, 1993.

[84] F. Mattern. "Virtual time and global states of distributed systems," Proc. Int. Workshop on Parallel and Distributed Algorithms Bonas, France, Oct. 1988, Cosnard, Quinton, Raynal, and Robert Eds., North Holland, 1989.

[85] S. Melzer, S. Römer, "Deadlock checking using net unfoldings," CAV'97, LNCS 1254, pp. 352-363.

[86] M. Nielsen, G. Plotkin, G. Winskel, "Petri nets, event structures and domains," Theoretical Computer Science 13(1), 1981, pp. 85-108.

[87] J. Pearl, "Fusion, Propagation, and Structuring in Belief Networks," Artificial Intelligence, vol. 29, pp. 241-288, 1986.

[88] Y. Pencole, M-O. Cordier, L. Roze, "A decentralized model-based diagnostic tool for complex systems," Int. J. on Artif. Intel. Tools, World Scientific Publishing Comp.

[89] G. Provan, "A model-based diagnosis framework for distributed systems," Int. Workshop on Principles of Diagnosis (DX'02), 2002.

[90] M. Raynal, "Distributed algorithms and protocols," Wiley & Sons, 1988

[91] W. Reisig, "Petri Nets," Springer Verlag, 1985.

[92] T. Richardson, R. Urbanke, "The Capacity of Low-Density Parity Check Codes Under Message-Passing Decoding," IEEE Trans. on Information Theory 47, pp. 599-618, Feb. 2001.

[93] T. Richardson, R. Urbanke, "Design of capacity-approaching low density parity-check codes," IEEE Trans. on Information Theory 47, pp. 619-637, Feb. 2001.

[94] T. Richardson, R. Urbanke, "An Introduction to the Analysis of Iterative Coding Systems," In Codes, Systems, and Graphical Models, IMA Volume in Mathematics and Its Applications, pages 1-37. Springer, 2001.

[95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D. Teneketzis, "Diagnosability of Discrete-event systems," IEEE Trans. Autom. Control, vol. 40(9), pp. 1555-1575, 1995.

[96] M. Sampath, R. Sengupta, K. Sinnamohideen, S. Lafortune, D. Teneketzis, "Failure diagnosis using discrete event models," IEEE Trans. on Systems Technology, vol. 4(2), pp. 105-124, March 1996.

[97] M. Steinder, A.S. Sethi, "End-to-End Service Failure Diagnosis Using Belief Networks," Network Operations and Management Symposium (NOMS), Florence, Italy, 2002.

[98] M. Steinder, A.S. Sethi, "Distributed Fault Localization in Hierarchically Routed Networks," IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'02), LNCS 2506, pp. 195-207, 2002.

[99] E. Sudderth, M. Wainwright, A. Willsky, "Embedded Trees: Estimation of Gaussian Processes on Graph with Cycles," IEEE Transactions on Signal Processing 52(11), Nov. 2004.

[100] E. Sudderth, A. Ihler, W. Freeman, A. Willsky, "Nonparametric Belief Propagation," IEEE Conference on Computer Vision and Pattern Recognition, June 2003.

[101] E. Sudderth, "Graphical Models for Visual Object Recognition and Tracking," PhD Thesis, Massachusetts Institute of Technology, May 2006.

[102] R. Su, "Distributed Diagnosis for Discrete-Event Systems," PhD Thesis, Dept. of Elec. and Comp. Eng., Univ. of Toronto, June 2004.

[103] R. Su, W.M. Wonham, J. Kurien, X. Koutsoukos, "Distributed Diagnosis for Qualitative Systems," in Proc. 6th Int. Workshop on Discrete Event Systems, WODES'02, pp. 169-174, 2002.

[104] S. ten Brink, "Convergence behaviour of iteratively decoded parallel concatenated codes," IEEE Trans. on Communications 49, Oct. 2001.

[105] S. Tripakis, "Undecidable Problems of Decentralized Observation and Control," in Proc. 40th Conf. on Decision and Control (CDC), 2001.

[106] F. W. Vaandrager, "A simple definition for parallel composition of prime events structures," Report CS-R8903, CWI, Amsterdam, March 1989.

[107] W. Vogler, A. Semenov, A. Yakovlev, "Unfolding and Finite Prefix for Nets with Read Arcs," 9th Int. Conf. on Concurrency Theory (CONCUR), LNCS 1466, pp. 501-516, 1998.

[108] Y. Weiss, W. T. Freeman, "On the Optimality of Solutions of the Max-Product Belief-Propagation Algorithm in Arbitrary Graphs," IEEE Trans. on Information Theory, vol. 47, no. 2, pp. 723-735, Feb. 2001.

[109] G. Winskel, "A new Definition of Morphism on Petri Nets," LNCS 166, pp. 140-149, 1984.

[110] G. Winskel, "Categories of models for concurrency," Seminar on Concurrency, Carnegie-Mellon Univ. (July 1984), LNCS 197, pp. 246-267, 1985.

[111] G. Winskel, "Event structure semantics of CCS and related languages," LNCS 140, 1982, also as report PB-159, Aarhus Univ., Denmark, April 1983.

[112] G. Winskel, "Petri Nets, Algebras, Morphisms, and Compositionality," Information and Computation, no. 72, pp. 197-238, 1997.

[113] J. S. Yedidia, W. T. Freeman, Y. Weiss, "Bethe free energy, Kikuchi approximations, and belief propagation algorithms," available at `www.merl.com/papers/TR2001-16/`

[114] J. S. Yedidia, W. T. Freeman, Y. Weiss, "Generalized Belief Propagation," Advances in Neural Information Processing Systems (NIPS), Vol 13, pp. 689-695, December 2000, also as MERL Tech. Rep. no. TR2000-26.

[115] T. Yoo, S. Lafortune, "A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems," J. Discrete Event Dynamic Systems: Theory and Applications, vol. 12(3), pp. 335-377, July, 2002.